



EnterpriseOne B73.3.1
Development Tools
Accounting PeopleBook

June 1999

J.D. Edwards World Source Company
One Technology Way
Denver, CO 80237

Portions of this document were reproduced from material prepared by J.D. Edwards.

Copyright ©J.D. Edwards World Source Company, 1997 - 1999

All Rights Reserved

SKU B7331CEAOD

J.D. Edwards is a registered trademark of J.D. Edwards & Company. The names of all other products and services of J.D. Edwards used herein are trademarks or registered trademarks of J.D. Edwards World Source Company.

All other product names used are trademarks or registered trademarks of their respective owners.

The information in this guide is confidential and a proprietary trade secret of J.D. Edwards World Source Company. It may not be copied, distributed, or disclosed without prior written permission. This guide is subject to change without notice and does not represent a commitment on the part of J.D. Edwards & Company and/or its subsidiaries. The software described in this guide is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. J.D. Edwards World Source Company uses automatic software disabling routines to monitor the license agreement. For more details about these routines, please refer to the technical product documentation.

OneWorld Development Tools Guide

The OneWorld *Development Tools* guide consists of several units, including.

- ☐ Fundamentals
- ☐ Application Design
- ☐ Event Rules
- ☐ Business Functions
- ☐ Caching
- ☐ Additional Features
- ☐ Batch Processing
- ☐ Messaging
- ☐ Debugging
- ☐ Web Applications
- ☐ Performance
- ☐ Appendices

Fundamentals

Fundamentals covers the basic concepts and tools you need to know to start developing applications. It includes the preliminary steps you must take before you actually design an application. Fundamentals includes the following sections:

- Object Librarian
- Data Dictionary
- Table Design
- Business View Design
- Data Structures
- Cross Reference

-
- Checkout Log

Application Design

Application Design includes the steps you follow to actually design an application. It describes different form types and controls. It includes the following sections:

- Application Design
- Forms Design

Event Rules

Event Rules discusses using event rules and logic. It also describes how to use BrowsER. It includes the following sections:

- Event Rules Design

Business Functions

Business Functions discusses both C business functions and named event rules. It also has chapters on master business functions, using Business Function Builder, and Business Function Documentation. It includes the following sections:

- Business Functions

Caching

Caching describes how you can use caching for better performance. It includes the following sections:

- Caching

Additional Features

Additional Features discusses additional features of the OneWorld development tools that you can use to enhance your applications. It includes the following sections:

- Processing Options
- Transaction Processing
- Currency
- Menu Design

Batch Processing

Batch Processing discusses the steps you take to design a batch application. It includes information about processing, events, level-breaks, and sections. This unit also includes information about subsystem jobs and database output. It includes the following sections:

- Batch Processing
- Database Output
- Subsystem Jobs

Messaging

Messaging discusses error handling, interactive messaging, and batch messaging. It includes the following sections:

- Messaging
- Batch Error Messages

Debugging

Debugging provides information to help you debug your applications. It includes different methods of debugging and strategies to help you. It includes the following sections:

- Debugging

Web Applications

Web Applications discusses designing and generating web applications. It includes the following sections:

- Developing Web Applications
- Generating Web Applications

Performance

Performance discusses troubleshooting your application and various performance issues. It includes tips to help you get better performance from your applications. It includes the following sections:

- Performance

Appendices

The appendices include information about making software modifications and detailed information about the process flow for different form types. This unit includes the following appendices:

- Modification Rules
- Form Processing
- Smart Fields
- Date Reference Scan

OneWorld Acronyms

Following are some acryonyms that are commonly used in OneWorld.

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 Rules
FDA	Form Design Aid
NER	Named Event Rules
OCM	Object Configuration Manager
OL	Object Librarian
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 Rules
UBE	Universal Batch Engine
WF	Work Flow

Table of Contents

OneWorld Tools	1-1
Understanding Objects and Applications	1-3
Understanding an Object	1-3
Understanding an Application	1-4
Understanding How to Build an Application	1-5
Understanding the Development Cycle	1-6
Understanding How OneWorld Stores Objects	1-7
Understanding OneWorld Tools	1-8
Object Librarian	1-8
Data Dictionary	1-8
Table Design	1-9
Business View Design	1-9
Form Design	1-9
Report Design	1-10
Business Function Design	1-10
Event Rules	1-10
Processing Options	1-11
Menus	1-12
How OneWorld Works at Design Time	1-13

Fundamentals

Object Librarian	2-1
Tracking Objects	2-2
Object Librarian Master Table	2-2
Object Librarian Detail Table	2-3
Object Librarian Function Detail	2-3
Object Librarian-Business Function Object Relationships	2-4
Form Information Table	2-5
Working with Object Librarian	2-7
Adding an Object	2-8
Understanding Object Types and Naming Conventions	2-12
Modifying an Object	2-17
Deleting an Object	2-21
Copying an Object	2-22
Working with Object Librarian Codes	2-23
Displaying a Reference List of Business Functions	2-28
Working with the Business Function Source Librarian	2-29
Working with Promotion Manager	2-33
Using Promotion Manager	2-33

Data Dictionary	3-1
How the Data Dictionary is Used at Runtime	3-2
Naming Data Items	3-3
Storing the Data Dictionary and Data Dictionary Items	3-3
Glossary Items	3-4
Error Messages	3-4
Understanding Default Triggers	3-4
Using the Data Dictionary	3-7
Defining a Data Item	3-11
Naming a Data Item	3-12
Data Dictionary Item Naming Conventions	3-12
Defining General Information	3-16
Attaching Default Triggers	3-23
Attaching a Default Value Trigger	3-25
Attaching a Visual Assist Trigger	3-26
Attaching an Edit Rule Trigger	3-28
Attaching a Display Rule Trigger	3-31
Attaching a User Defined Codes Trigger	3-33
Attaching a Next Number Trigger	3-36
Attaching a Smart Field Trigger	3-38
Updating the Glossary	3-40
Table Design	4-1
Adding a Table	4-3
Indices	4-5
Working with Table Design	4-9
Choosing Data Items for the Table	4-10
Defining Indices	4-11
Generating Tables	4-13
Previewing Tables	4-17
Copying Tables	4-18
See Also	4-19
Viewing the Data in Tables	4-21
Example: Universal Table Browser (Unformatted Data)	4-24
Example: Universal Table Browser (formatted data)	4-25
Example: Column Properties	4-26
Business View Design	5-1
Table Join	5-1
Table Union	5-2
Select Distinct	5-2
Indices	5-2
Adding a Business View	5-3
Joined Views	5-5
Working with Business View Design	5-7
Choosing a Table for a Business View	5-8
Choosing Data Items for a Business View	5-9
Using Select Distinct	5-10
Creating a Table Join	5-14
Creating a Table Union	5-16
Generating a Business View	5-16

Data Structures	6-1
System Generated Data Structures	6-1
User Generated Data Structures	6-1
Business Function Data Structures	6-2
Working with Interconnection Data Structures	6-3
Modifying Form Data Structures	6-3
Modifying Report Data Structures	6-5
Example: Changing Interconnection Data Structures	6-6
Displaying Type Definitions	6-7
Creating a Data Structure	6-9
Creating a Media Object Data Structure	6-9
Creating a Processing Options Data Structure	6-11
Creating a Business Function Data Structure	6-12
 Cross Reference Facility	 7-1
Working with the Cross Reference Facility	7-3
Searching for Objects	7-3
Searching for Data Items	7-4
Searching for Interactive Applications	7-5
Searching for Batch Applications	7-6
Searching for Business Functions	7-7
Searching for Business Views	7-8
Searching for Data Structures	7-9
Searching for Tables	7-10
Searching for Forms	7-11
Searching for Event Rules	7-12
Viewing Field Relationships	7-14
Rebuilding Cross Reference Information	7-15
 Checkout Log	 8-1
Working with the Checkout Log	8-3
Confirming That an Object Has Been Checked In	8-3
Purging the Checkout Log	8-5

Application Design

Application Design	9-1
Understanding Application Design	9-3
Understanding Interactive Applications	9-3
Understanding Batch Applications	9-4
Understanding Web Applications	9-4
Adding an Interactive Application	9-5
 Forms Design	 10-1
What is a Form?	10-2
Elements of a Form	10-3
Understanding Form Types	10-5
About J.D. Edwards Design Standards	10-5
Find/Browse Forms	10-6

Fix/Inspect Forms	10-7
Header Detail Forms	10-9
Headerless Detail Forms	10-10
Search and Select Forms	10-12
Message Forms	10-13
Parent/Child Forms	10-14
Creating a Form	10-17
Forms	10-20
Designing a Form Layout	10-25
Selecting and Moving Controls	10-25
Changing the Size of Grids, and Controls	10-26
Using Cut, Copy, and Paste in Forms	10-29
Aligning Controls	10-29
Using Undo and Redo	10-32
Creating Menu/Toolbar Exits	10-33
Adding Bitmaps	10-36
Attaching Event Rules to a Menu/Toolbar Exit	10-38
Creating Subcategories in Menus	10-40
Working with Controls	10-43
Understanding Form Controls	10-44
Creating Push Buttons	10-45
Creating Check Boxes	10-46
Creating Radio Buttons	10-48
Adding Data Items to a Form	10-50
Inserting Database Items Onto a Form	10-50
Inserting Data Dictionary Items Onto a Form	10-51
Creating Static Text Controls	10-52
Creating Edit Controls	10-54
Creating UDC Edit Controls	10-58
Disconnecting Static Text from Controls	10-60
Creating Media Object Controls	10-60
Creating Bitmap Controls	10-66
Creating a Tree Control	10-68
Creating Group Boxes	10-70
Defining Grid Properties	10-71
Defining Grid Column Properties	10-74
Defining Properties for Parent/Child Controls	10-76
Defining Options for Forms, Grid, and Edit Controls	10-87
Defining Options for a Form	10-87
Defining Options For a Grid	10-88
Defining Options For an Edit or UDC Edit Control	10-90
Associating Database Items, Dictionary Items, or Descriptions	10-91
Database Items	10-91
Dictionary Items	10-91
Descriptions	10-91
Changing Tab Sequence	10-92
Creating Tab Controls	10-94
Overriding Data Dictionary Triggers at Design Time	10-97
Designing Forms Using Multiple Modes	10-107
Viewing Forms in a Particular Mode	10-107
Setting Control Mode Properties	10-108

Using Text Variables	10-111
Adding a Text Variable	10-111
Attaching or Assigning a Text Variable	10-112
Using Quick Form	10-115
Processing Media Objects	10-117
Imaging	10-117
Using Standard Processing for Media Objects	10-118
Language Considerations for Media Objects	10-121
Testing a Form	10-127
Language Preview	10-128

Event Rules

Event Rules Design	11-1
Understanding Controls	11-2
Understanding Events	11-2
Understanding Form Processing	11-2
Understanding Event Rules	11-3
Business Function Event Rules	11-3
Embedded Event Rules	11-3
Application Event Rules	11-4
Table Event Rules	11-4
Understanding the Event Rule Buttons	11-4
Event Rules Based on Form Type	11-5
Find/Browse	11-5
Form-Level Events	11-5
Grid-Level Events	11-5
Fix/Inspect	11-5
Header Detail and Headerless Detail	11-6
Form-Level Events	11-6
Grid-Level Events	11-7
Runtime Processing	11-9
Runtime Structures	11-9
Available Objects and Runtime Data Structures	11-9
Event Rule Manipulation	11-12
Processing Available Objects	11-12
Control is Exited Processing	11-12
Grid Processing	11-13
Form Flow	11-14
Typical Event Flow for a Find/Browse Form	11-14
Pre Dialog Is Initialized	11-15
Dialog Is Initialized	11-16
Post Dialog Is Initialized	11-17
SQL SELECT	11-18
Do in While Loop	11-18
Page-At-A-Time Processing	11-18
BC Assigned Database Values	11-19
Grid Rec Is Fetched	11-19
Write Grid Line Before	11-21

Write Grid Line After	11-23
Last Grid Rec Has Been Read	11-24
Select Button Processing	11-25
Button Clicked	11-26
Add Button Processing	11-27
Delete Button Processing	11-28
Delete Grid Rec Verify Before	11-28
Delete Grid Rec Verify After	11-29
Delete Grid Rec From DB Before	11-29
Delete Grid Rec From DB After	11-30
All Grid Recs Deleted From DB	11-30
Working with Event Rules Design	11-31
Creating and Saving Event Rules	11-31
Finding Event Rules	11-34
Cutting, Copying, and Pasting Event Rules	11-35
Adding Comment Lines to Event Rules	11-39
Printing Event Rule Logic	11-40
Validating Event Rules	11-42
Working with If and While Statements	11-45
Creating If and While Statements	11-46
Defining Literal Values in Event Rule Logic	11-50
Working with Event Rule Variables	11-53
Interactive Event Rule Variables	11-53
Batch Application Event Rule Variables	11-54
Creating an Event Rule Variable	11-54
J.D. Edwards Naming Standards	11-58
Using Event Rule Variables for Automatic Line Numbering	11-59
Attaching Functions	11-61
Attaching a System Function to an Event	11-61
Common System Functions	11-63
Control System Functions	11-63
Message System Functions	11-64
General System Functions	11-65
Grid System Functions	11-66
Telephony System Functions	11-68
Attaching a Business Function to an Event	11-69
Creating Form Interconnections	11-75
Modal Processing	11-75
Modeless Processing	11-75
Creating a Modal Form Interconnection	11-76
Creating a Modeless Form Interconnection	11-80
Creating Report Interconnections	11-85
Creating a Report Interconnection	11-85
Creating Assignments	11-89
Expression Manager	11-92
Table I/O	11-97
Available Operations	11-97
Valid Mapping Operators	11-98
Creating a Table I/O Event Rule	11-98
Understanding Handles	11-102

Creating a Handle	11-102
Opening Handles	11-105
Table Event Rules	11-107
Creating Table Event Rules	11-107
Creating Dynamic Overrides	11-113
Working with Asynchronous Processing	11-115
Asynchronous Events	11-116
Control is Exited and Changed - Asynch	11-117
Row is Exited and Changed - Asynch	11-117
Column is Exited and Changed - Asynch	11-117
Asynchronous Business Functions	11-118
Example: Asynchronous Event Processing	11-119
Working with BrowsER	11-121
Working with BrowsER	11-121

Business Functions

Business Functions	12-1
Business Function Features	12-2
What are the Components of a Business Function?	12-3
How Distributed Business Functions Work	12-5
Creating Business Function Event Rules	12-7
Understanding C Business Functions	12-15
Header File Sections	12-16
Business Function Header File Example	12-17
What is the Structure of a Business Function Source File?	12-20
Example: Business Function Source File Check for In Add Mode ..	12-20
Using Application Programming Interfaces (APIs)	12-23
OneWorld-Specific Data Types	12-23
MATH_NUMERIC Data Type	12-24
JDEDATE Data Type	12-24
Using Database APIs	12-25
Understanding Standards and Portability	12-25
J.D. Edwards Open Database Connectivity (ODBC)	12-26
Standard JDEBASE API Categories	12-27
Connecting to a Database	12-28
Understanding Database Communication Steps	12-28
Calling an API from an External Business Function	12-29
Calling a Visual Basic Program from OneWorld	12-31
Working with Business Functions	12-35
Viewing Object Codes	12-35
Adding Related Tables and Functions	12-36
Debugging Business Functions	12-38
Changing a Business Function Parent DLL or Location	12-39
Creating and Specifying Custom DLLs	12-41
Creating a Custom DLL	12-41
Specifying a Custom DLL for a Custom Business Function	12-43
Working with Business Function Builder	12-45
Understanding the Business Function Builder Form	12-45

Functions List	12-46
Available Business Functions	12-46
Build Output	12-46
Building a Single Business Function	12-47
Linking Business Function Objects	12-49
Setting Build Options	12-49
Using Utility Programs	12-51
Reading Build Output	12-51
Makefile Section	12-52
Begin DLL Section	12-52
Compile Section	12-52
Link Section	12-53
Rebase Section	12-53
Summary Section	12-54
Building All Business Functions	12-54
Reviewing Error Output	12-61
Resolving Errors	12-62
Transaction Master Business Functions	12-65
Master Business Function Naming Convention	12-65
Creating Processing Options for a Master Business Function	12-66
Naming Transaction Master Business Function Modules	12-66
Components of a Transaction Master Business Function	12-67
Begin Document	12-67
Edit Line	12-70
Edit Document	12-72
End Document	12-73
Clear Cache	12-74
Cancel Document (optional)	12-75
Cache Structure	12-75
Building Transaction Master Business Functions	12-78
Implementing Transaction Master Business Functions	12-79
Single Record Processing	12-79
Document Processing	12-80
Master File Master Business Functions	12-81
Information Structure	12-83
Performance Impact	12-84
Business Function Documentation	12-87
Creating Business Function Documentation	12-87
Generating Business Function Documentation	12-95
Viewing Business Function Documentation	12-98

Caching

Caching	13-1
JDECACHE	13-2
Performance Considerations	13-2
The JDECACHE API Set	13-3
JDECACHE Management APIs	13-3
JDECACHE Manipulation APIs	13-4

Working with JDECACHE	13-5
Calling JDECACHE APIs	13-5
Setting Up Indices	13-6
Initializing the Cache	13-7
Using an Index to Access the Cache	13-9
Using the jdeCacheInit/jdeCacheTerminate Rule	13-12
Using the Same Cache in Multiple Business Functions or Forms	13-13
JDECACHE Cursors	13-15
Opening a JDECACHE Cursor	13-15
HJDECURSOR	13-15
JDECACHE Dataset	13-16
Types of JDECACHE Cursors APIs	13-17
Updating Records	13-18
Deleting Records	13-19
The jdeCacheFetchPosition API	13-19
Resetting the Cursor	13-19
Closing a Cursor	13-19
JDECACHE Multiple Cursor Support	13-20
JDECACHE Partial Keys	13-21
How a JDECACHE Partial Key Works	13-21
JDECACHE Errors	13-23
JDECACHE Error References	13-23
JDECACHE Example Program	13-25
JDECACHE Standards	13-35
Cache Programming Standards	13-36
Individual Cache Business Function Header File	13-41

Additional Features

Processing Options	14-1
Processing Options Templates	14-2
Creating a Processing Options Data Structure	14-5
Processing Option Data Structure	14-10
Tab Title	14-11
Comment	14-12
Data Item	14-12
Attaching a Processing Options Template	14-17
Transaction Processing	15-1
Commits and Rollbacks	15-1
Understanding OneWorld Transaction Processing	15-3
Transaction Processing Scenarios	15-4
Transaction Processing and Business Functions	15-5
Transaction Processing in Remote Business Functions	15-6
Transaction Processing System Functions	15-6
Working with Transaction Processing	15-7
Defining Transaction Processing for a Form	15-8
Extending a Transaction Boundary	15-9
Defining Transaction Processing for a Report	15-19

Understanding Record Locking	15–21
Optimistic Locking	15–21
Pessimistic Locking	15–22
Currency	16–1
OneWorld Currency Implementation	16–1
Advantages	16–1
Working with Currency	16–3
Build Triggers Option	16–3
How Table Event Rules Work with Currency Processing:	16–4
Setting Up Currency Conversion	16–5
Showing Currency-Sensitive Controls	16–6
Creating a Currency Conversion Trigger	16–7
Menu Design	17–1
Understanding Menus	17–1
Menu Filtering	17–1
Menu Design Tables	17–2
Working with Menus	17–3
Defining a New Menu	17–3
Reviewing Selections for a Menu	17–6
Printing a Menu Report	17–6
Example: Print Menus Report	17–7
Working with Menu Selections	17–9
Adding or Changing a Menu Selection	17–9
Adding an Application to a Menu	17–13
Adding or Changing Web Addresses on OneWorld Explorer Help	17–20
Creating a Web View Subheading on a Menu	17–21
Linking Menus	17–21
Creating Fast Path Selections	17–22
Working with Menu Selection Revisions	17–25
Copying a Menu Selection	17–25
Changing Menu Text for Languages	17–26
Changing Menu Selection Text	17–28
Renumbering a Menu Selection	17–28

Batch Processing

Batch Processing	18–1
Understanding Batch Processing	18–3
Report Sections	18–3
Understanding Section Processing	18–9
Processing Group and Columnar Sections	18–9
Processing Tabular Sections	18–11
Attaching Logic to Joined Sections	18–12
Data Selection and Sequencing	18–13
How OneWorld Allocates Memory for Sections	18–13
Understanding Custom Sections	18–15
Attaching Logic for a Custom Section	18–16

Using a Custom Section	18-17
Understanding Level-Break Processing	18-19
Level-Break Header Sections	18-21
Using Level-Break Headers with Page Breaks	18-21
Level-Break Footer Sections	18-21
Conditionalizing Level-Break Headers or Footers	18-21
Level-Break Section Events	18-21
INIT_Level Break Footer Section Event	18-22
INIT_Level Break Header Section Event	18-22
END_Level-Break Header and Footer Section Events	18-22
Understanding Batch Events	18-23
The Do_Section Event	18-23
Additional Batch Events	18-23
Section Level Events	18-24
Page Header Section Level Events	18-25
Page Footer Section Level Events	18-26
Report Header Section Level Events	18-26
Report Footer Section Level Events	18-26
Variable Events	18-27
Variable Population at Specific Events	18-27
System Functions within Batch Events	18-31
Understanding Batch Runtime Processing	18-33
Available Objects	18-33
Processing Available Batch Objects	18-34
Typical Event Flow for a Group Section	18-34
Database Output	19-1
Defining Database Output	19-3
Overriding Environments for Database Output	19-7
Subsystem Jobs	20-1
Defining Subsystem Jobs	20-3
Adding an API Record to the Subsystem Table	20-7
Example: Subsystem Job Header File	20-7

Messaging

Messaging	21-1
Message Types	21-1
Error Messages	21-2
Workflow Messages	21-2
Level Messages	21-2
Information Messages	21-3
Understanding Error Handling	21-5
Event-Driven Model	21-5
When is an Error Set?	21-6
How Does the Application Know Which Field to Highlight?	21-6
How Is an Event ID Used for Error Setting?	21-6
Error Setting	21-7

Automatic Error Setting	21-7
Manual Error Setting	21-7
Business Function API	21-8
Resetting Errors	21-9
Resetting Errors on the OK Button	21-9
Resetting Errors on the Find Button	21-10
Multilevel Error Messaging for C Business Functions	21-11
Working with Error Messages	21-13
Locating an Existing Message	21-13
Creating a Simple Error Message	21-14
Adding an Interactive Message Data Item	21-15
Creating a Text Substitution Error Message	21-16
Adding an Interactive Message Data Item	21-17
Defining the Glossary Text for a Runtime Message	21-19
Attaching a Data Structure Template to a Message	21-20
Attaching an Interactive Message Data Item	21-21
Working with the Send Message System Function	21-25
Batch Error Messages	22-1
Understanding Batch Error Messaging	22-3
Level-Break Messages	22-3
Text Substitution Error Messages	22-5
Action Messages	22-5
Understanding How Level Break Messages Work	22-6
Work Center API	22-7
Creating a Level-Break Message	22-9
Creating a Data Dictionary Item for a Level-Break Message	22-9
Creating a Data Structure for the Data Dictionary Item	22-12
Creating a Level-Break Message Business Function Data Structure	22-13
Creating a Level-Break Business Function	22-15
Sample Source Code Highlights	22-15
Calling the Work Center Initialization API	22-18
Calling the Processing Work Center APIs	22-20
Terminating the Work Center Process	22-23

Debugging

Debugging	23-1
Overview of the Debugging Process	23-1
Interpretive and Compiled Code	23-3
Working with the Event Rules Debugger	23-5
Enabling Debug Information	23-5
Understanding the Event Rules Debugger	23-7
Application Tree Control	23-8
Event Rules Window	23-8
Variable Selection and Display Control	23-9
Breakpoint Manager	23-9
Search Combo Box	23-10
Debugging an Application	23-11

Inspecting or Modifying a Variable	23-12
Just in Time Debugging	23-13
Setting Breakpoints	23-14
Debugging Business Functions Using Microsoft Visual C++	23-15
Working with the Visual C++ Debugger	23-21
Useful Features of the Visual C++ Debugger	23-21
The Go Command	23-21
The Step Command	23-21
The Step Into Command	23-21
Setting Breakpoints	23-22
Using Watch	23-22
Locals Window	23-22
Example: Debugging with the Visual C++ Debugger	23-22
Customizing the Environment	23-23
Debugging Strategies	23-25
Is the Program Ending Unexpectedly?	23-25
Is the Application Encountering Errors?	23-26
Is the Output of the Program Incorrect?	23-26
Where Could the Problem Be Coming From?	23-26
Is the Function Being Called as Expected?	23-26
Debug Logs	23-27
SQL Log Tracing	23-28
Debug Tracing	23-29
Turning on Debug Tracing	23-29
System Function Tracing	23-29

Web Applications

Developing Web Applications	24-1
Creating Web Applications	24-4
Generating Web Applications	24-7
Generating a Form	24-9
Generating an Application	24-14
Generating all Applications	24-15

Performance

Performance	25-1
Performance Tips	25-3
Things to Look For	25-3
Data Dictionary Performance	25-4
Triggers	25-4
Overrides	25-4
Validation	25-5
Table Design Performance	25-5
Index Limitations for Various Databases	25-6
Access32	25-6
SQLSERVER	25-6

DB2 for OS/400	25-6
Oracle	25-7
Key Column Violation	25-7
Specification File Corruption	25-8
Table I/O Objects	25-8
Business View Performance	25-9
Data Structure Performance	25-10
Data Selection and Sequencing	25-10
Form Design	25-11
Find/Browse	25-11
Header Detail and Headerless Detail	25-11
Batch Application Performance	25-12
Event Rules Performance	25-12
Business Function Performance	25-14
Error Messaging Performance	25-15
Transaction Processing Performance	25-15
Troubleshooting	25-15

Appendices

Appendix A: OneWorld Modification Rules	A-1
What an Upgrade Preserves and Replaces	A-3
Understanding Interactive Application Rules	A-4
Understanding Report Rules	A-5
Understanding Application Text Changes	A-6
Understanding Table Specification Rules	A-6
Understanding Control Table Rules	A-7
Understanding Business Views Rules	A-7
Understanding Rules for Event Rules	A-8
Understanding Data Structure Rules	A-8
Understanding Business Functions Rules	A-9
Understanding Version Rules	A-10
Appendix B: Form Processing	B-1
Process Flow for Find/Browse Form	B-3
Process Flow for Parent/Child Form	B-9
Process Flow for Fix/Inspect Form	B-15
Process Flow for Header Detail Form	B-19
Process Flow for Headerless Detail Form	B-27
Process Flow for Search/Select Form	B-35
Process Flow for Message Form	B-39
Process Flow for Edit Control	B-41
Process Flow for Grid Control	B-45
Appendix C: Smart Fields	C-1
Understanding Smart Fields	C-3
Creating Smart Fields	C-7
Creating a Named Mapping	C-7

Appendix D: Date Reference Scan D-1
Working with the Date Reference Scan D-3
 Business Function Date Reference Scan D-3
 Event Rule Date Reference Scan D-6

Glossary

Index



OneWorld Tools

The OneWorld Tools are an integrated set of application development tools. These tools allow business analysts to develop complete interactive or batch applications (for example forms and reports). The tools simplify the development process and limit the amount of programming necessary to create applications.

OneWorld Tools also allow you to create applications for client/server environments using the stability of J.D. Edwards methodology and the ease of the Microsoft Windows interface.

This section describes:

- ☐ Understanding objects and applications
- ☐ Understanding how to build an application



Understanding Objects and Applications

J.D. Edwards uses OneWorld Tools to build objects, which are used to build applications.

This chapter describes the following:

- ☐ Understanding an object
- ☐ Understanding an application

Understanding an Object

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

A specification is a complete description of a OneWorld object. Each object has its own specification, which is stored on both the server and the workstation. Some specifications describe different types of objects. For example, the data structure specification is used to describe a business function data structure, a processing option structure and a media object structure.

The OneWorld architecture is object-based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. It is this use of objects (applications being broken down into smaller components) that allows J.D. Edwards to provide true distributed processing. Developers create the objects using OneWorld Tools.

Examples of OneWorld objects include:

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

- Tables

Understanding an Application

An application is a collection of objects that performs a specific task. J.D. Edwards uses the OneWorld Tools to build its standard groups of related applications:

- Architecture, engineering, and construction
- Distribution
- Energy and chemical systems
- Financial
- Human resources
- Manufacturing
- Technical

These applications share a common user interface because they are all generated through OneWorld Tools. Applications refer to both interactive and batch applications. For example, all of the following are applications:

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

Understanding How to Build an Application

You can use the OneWorld Tools to build your applications. This chapter briefly describes the development cycle and each OneWorld tool.

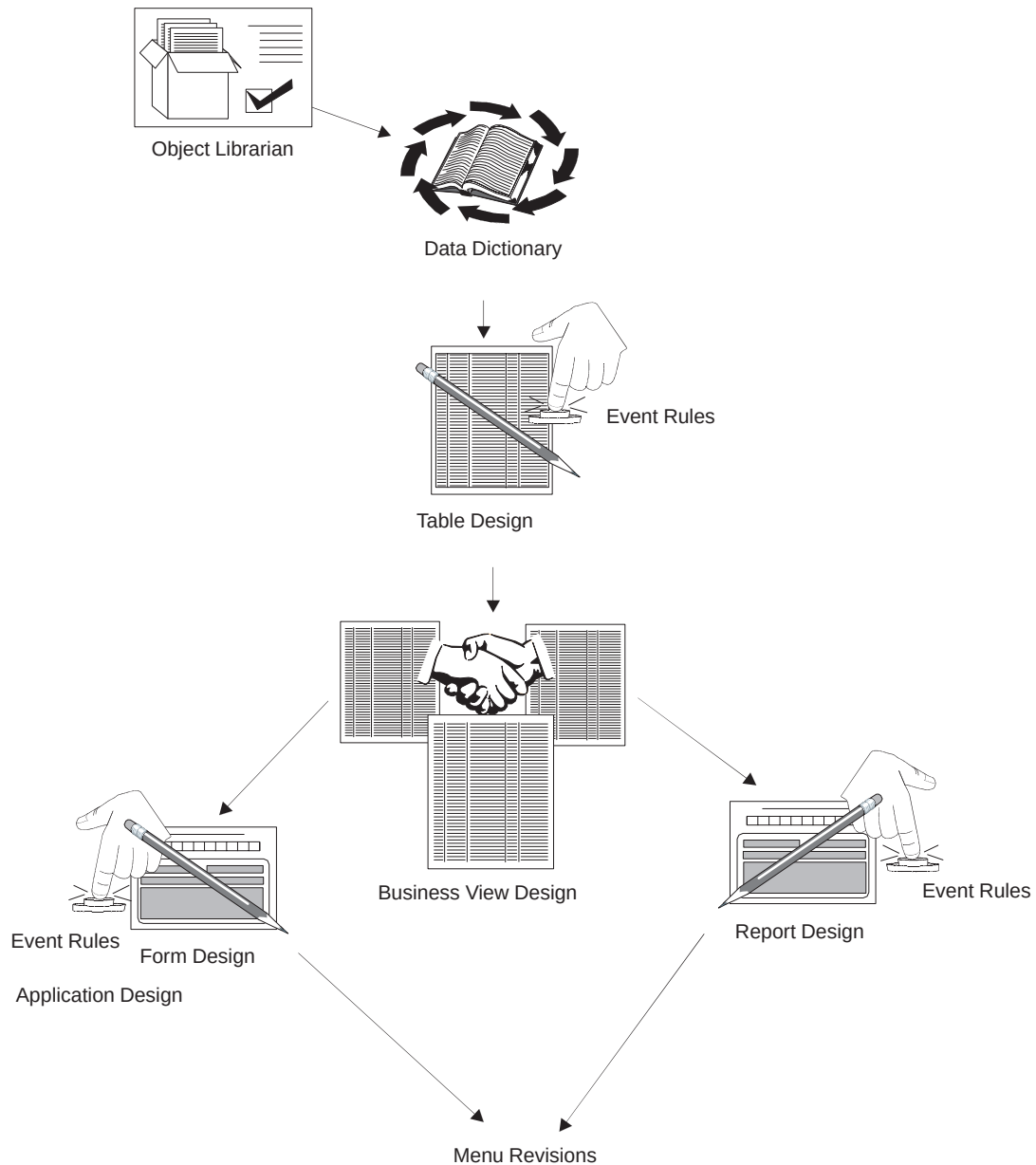
You might not need to use every tool to create an application; however, you always begin your application development from the Object Librarian. For example, you might not need to add or modify data items. If so, you can proceed to Table Design from the Object Librarian. If one or more existing database tables already contain all of the data items you want in your application, then you can skip the step of designing a table and proceed to Business View Design.

This chapter describes the following:

- ☐ Understanding the development cycle
- ☐ Understanding how OneWorld stores objects
- ☐ Understanding OneWorld Tools

Understanding the Development Cycle

The following figure illustrates the software development cycle, showing relationships between the tools.



Understanding How OneWorld Stores Objects

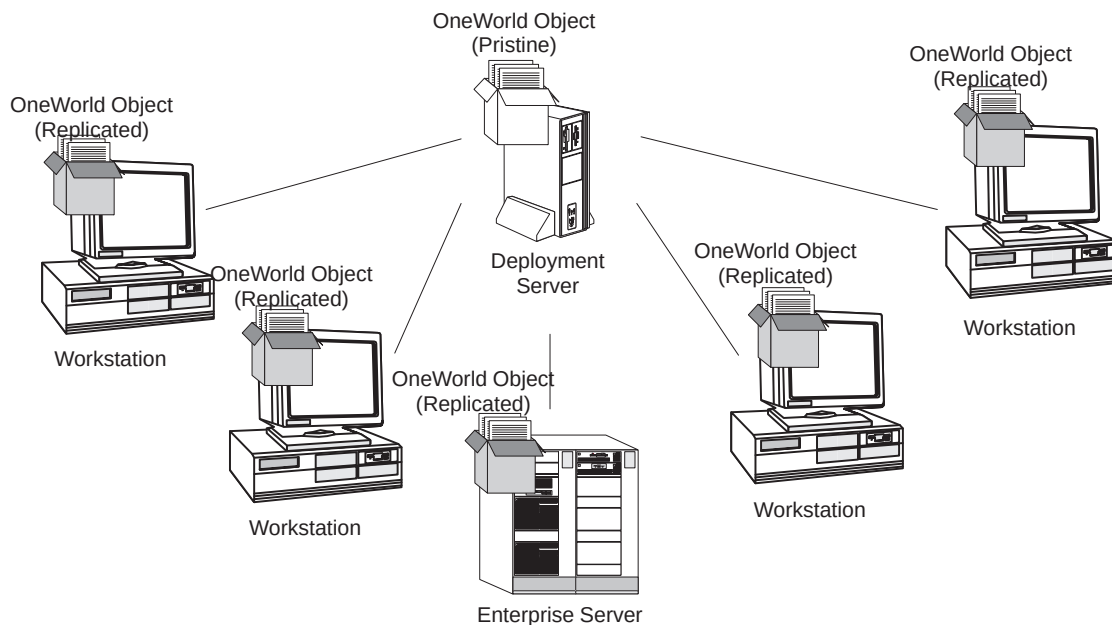
OneWorld stores objects in two places:

- A “central-storage” server, which stores central objects.

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

- On any machine (workstation or server) that runs OneWorld and stores replicated objects.

A copy (replicated) set of the central objects must reside on each workstation and server that run OneWorld. The path code indicates the directory where these objects are located.



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

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

Understanding OneWorld Tools

The OneWorld Tool Set is comprised of several tools that you use to create an application or report:

- Object Librarian
- Data Dictionary
- Table Design
- Business View Design
- Form Design
- Report Design
- Business Function Design
- Event Rules
- Processing Options
- Menus

Object Librarian

The OneWorld Object Librarian manages all objects. Developers use the Object Librarian to check out objects from a central development environment, copying the objects to their desktop. They can then use the tools to change objects and check the objects back in for others to access. Developers can access only OneWorld Tools through the Object Librarian.

Data Dictionary

Just as a dictionary contains word definitions, the J.D. Edwards data dictionary is a central repository that contains data item definitions and attributes. These attributes determine how a data item:

- Appears on reports and forms (such as number of decimals and default values)
- 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 changes are automatically reflected in applications without having to recompile software.

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, select data items for a table, and assign key fields as indices for retrieving and updating data.

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, select the data items from one or more tables that you want in your business view. With business views, you can select only the columns needed in the application, which increases performance due to less data moving over the network. For example, you could create a business view that contains only employee names and addresses from a table containing all employee data.

Form Design

An interactive application is the bridge between a user and a database (based on a business view). A user accesses an interactive application to add, modify, or view data using a form.

To create an application, determine the type of form 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.

There are several standard form types that 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, such as Sales Order Header
Header Detail	Used to modify multiple records at a time (particularly on normalized tables), such as Sales Order Detail
Headerless Detail	Used to modify multiple records in a table that is not normalized, such as Voucher Entry or Bill of Material
Search and Select	Used to automatically retrieve data into a visual assist field
Parent/Child	Used to display multiple records that have a parent/child relationship in an Explorer-like tree view, for example Bill of Material

Message

Used to display messages or request action, for example
Delete Confirmation

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, determine the data you want displayed. 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. This code can be written in most industry standard third-generation languages. You use Business Function Design to write business functions to provide background processing to handle specialized tasks needed in an application, such as specialized editing on 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 on a report. Event rules process 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 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 upon completion of an interactive application
- Attach a business function or system function
- Initiate workflow processes

There are two types of Event Rules:

- 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. They 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 re-useable. Examples include form-to-form calls, hiding a field based on a processing options value, and calling a business function. You can have embedded event rules in application event rules (interactive or batch) or in table event rules.

- Application Event Rules

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

- Table Event Rules

You can create OneWorld 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, you might have rules on a master table to delete all children when a parent is deleted, which maintains referential integrity. Any application that initiates a delete of that table does not need the parent/child logic imbedded in the application because it resides at the table level.

Processing Options

Processing options control how interactive and batch applications process data. You can have several versions of the same basic set of processing options. Each version

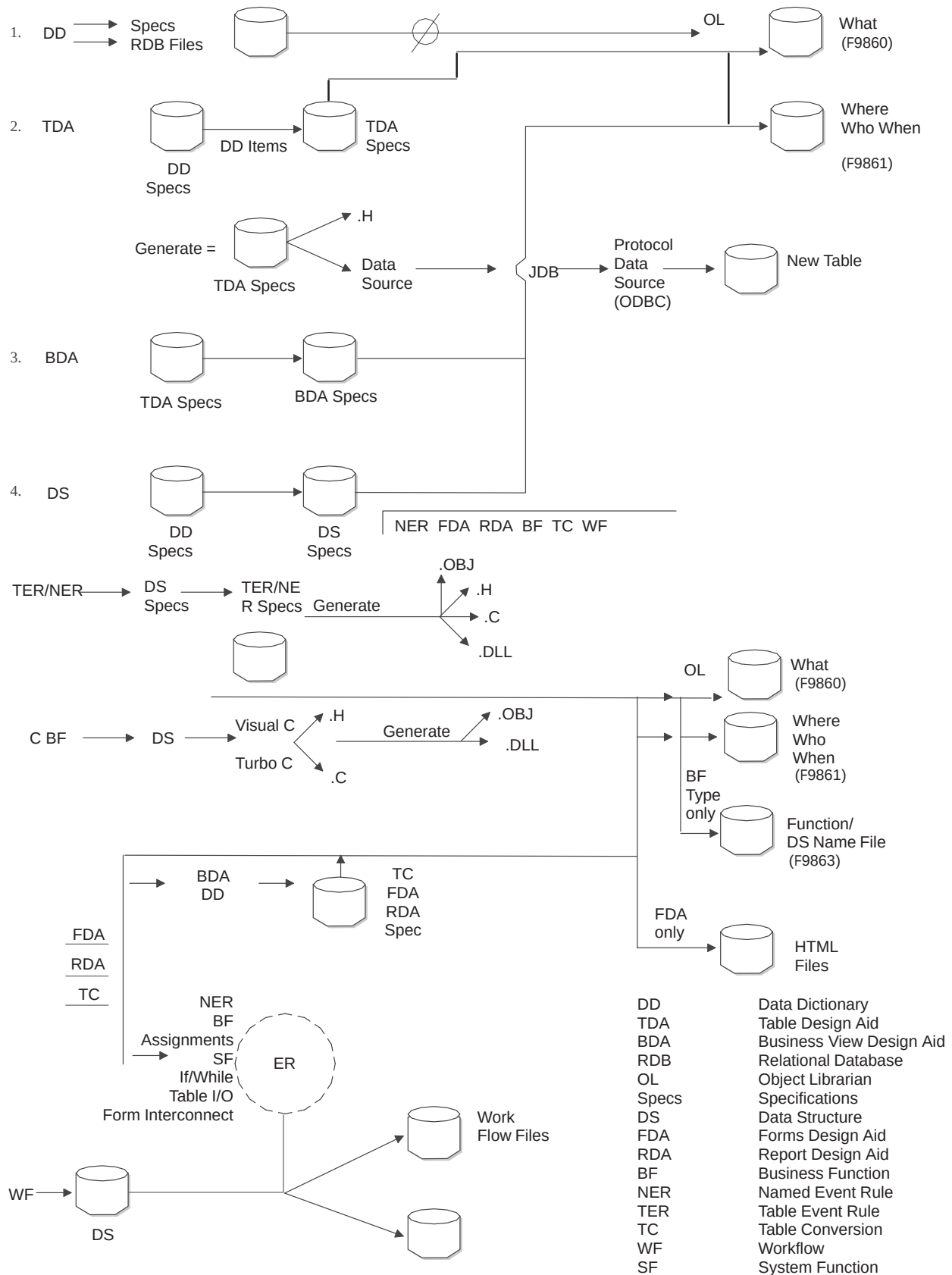
might have one slight difference from another version. For example, you can use processing options to do the following:

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

Menus

Menus are the entry point to J.D. Edwards applications and reports. To access an application or report from a menu, it must be attached to a menu selection, and menu selections must be attached to a menu. See the *OneWorld Foundation* guide for information about creating customized menus.

How OneWorld Works at Design Time



Fundamentals



Object Librarian

An application is composed of many objects. You can use the OneWorld Development Tools to create objects. Objects are managed by Object Librarian.

The Object Librarian:

- Is the entry point to the tool set
- Manages the creation and modification of objects
- Tracks the location of objects
- Identifies which developers are using objects
- Runs applications
- Displays a reference list of business functions

This section describes the following:

- ☐ Working with Object Librarian



Tracking Objects

The Object Librarian works with the OneWorld Development Tools to control the movement of objects between the client and the server.

When you create an object, specification records exist only on your workstation. To make the object available to other users, you must check in the object. Checking in an object copies the specification records to the server using the path code you select. To modify an existing object, you must first check out the object. Checking out an object copies the specification records from the server to your workstation.

More than one user can check out most objects. If more than one user has an object checked out, the Object Librarian displays a list of the users that have checked out the object. Only one user at a time can check out a business function. You can use processing options to allow only one check out at a time for other object types.

If a user has an object checked out in Vocabulary Overrides, other users can erase their check out for that object (a warning will appear), but they cannot perform the following actions for that object:

- Check out
- Check in
- Delete
- Force a check out

The server line for the object will have a status code of “5” to indicate the object is in use by vocabulary overrides.

The Object Librarian uses the following tables to track information for each object:

- Object Librarian Master Table - F9860
- Object Librarian Detail Table - F9861
- Object Librarian Function Detail - F9862
- Object Librarian-Business Function Object Relationships - F9863
- Form Information File - F9865

Object Librarian Master Table

The Object Librarian Master table (F9860) contains a header record for each OneWorld object. When an object is added, the Object Librarian creates a new header record in the Object Librarian Master Table on the server. Each record is keyed on the object name. The Object Librarian Master Table includes the following information for each header record:

- Object Name

-
- Description
 - System Code
 - Reporting System Code
 - Object Type
 - File Prefix (for tables)
 - Library Parent DLL (for business functions)
 - User
 - Workstation ID
 - Date Modified
 - Time Modified

Object Librarian Detail Table

The Object Librarian Detail table (F9861) tracks current users of an object and provides information about whether the object is checked out to any workstation or user and the server path code in which the object exists. When an object is checked out, the Object Librarian creates a detail record for the workstation or user that includes:

- Client/Server ID
- Path Code
- Status Code
- Release Code
- SAR Number
- Development Progress Code
- User
- Workstation ID
- Date Modified
- Time Modified

When you check in or check out an object, only specification records that have changed are copied.

Object Librarian Function Detail

The Object Librarian Function Detail table (F9862) contains a record for each business function in a business function module. The Object Librarian Function Detail table links each name of a business function module to the function constituents. The table is keyed on the individual business function names.

When you add a function to a business function object, the Object Librarian creates a record in the Object Librarian Function Detail table for each function in the business function object. This record includes:

- Object Name
- Function Name
- Description
- Business Function Categories
- User
- Workstation ID
- Program ID
- Date Modified
- Time Modified
- EventSpecKey (for named event rules only)

Object Librarian-Business Function Object Relationships

The Object Librarian - Business Function Object Relationships table (F9863) stores relationships across business functions and relationships between a business function and a table.

The Object Librarian - Business Function Object Relationships table does not have anything to do with related tables within an application. It pertains strictly to relationships for C business functions. It is used when building a business function to reference the headers to which this object must be linked.

When you create a business function object, the Object Librarian also creates a record in the Object Librarian - Business Function Object Relationships table for each related object that is required for the source file. For example, if a table is referenced in functions within the source file, then it is recorded in the Object Librarian - Business Function Object Relationships table. The record includes:

- Object Name
- Object Type
- Release Code - Object
- User
- Program ID
- Date Modified
- Time Modified

Form Information Table

The Form Information table (F9865) provides a reference of all forms used within an application. It is updated when you check in an application. When an application is checked in, a record is created in the Form Information table that includes:

- Form Name
- Form ID
- Description
- Process Type
- Release Code
- Entry Point Indicator
- System Code
- Object Type
- Object Name
- Application ID
- Help File ID
- Help File Name
- User ID
- Program ID
- Date Modified
- Time Modified

The Form Information table is used to retrieve the help file name and ID for online help.

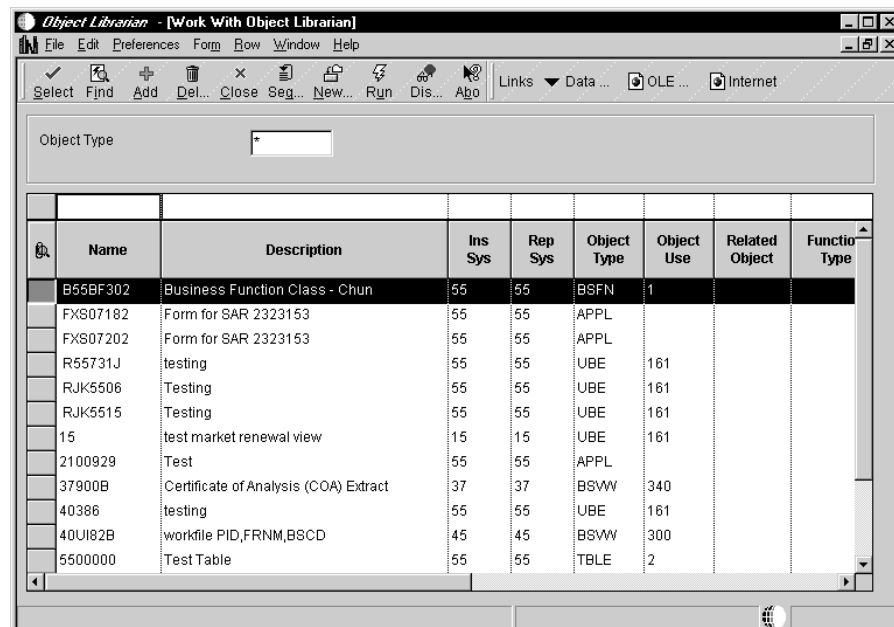
Working with Object Librarian

The Object Librarian accesses OneWorld Tools for adding a new object or modifying an existing one.

Using the OneWorld Explorer, from the Cross Application Development Tools menu (GH902), double-click the Object Librarian application (P9860).

This chapter describes the following:

- ☐ Adding an object
- ☐ Understanding Object Types and Naming Conventions
- ☐ Modifying an object
- ☐ Working with Object Librarian Codes
- ☐ Displaying a reference list of business functions



The screenshot shows the 'Object Librarian - [Work With Object Librarian]' window. It features a menu bar (File, Edit, Preferences, Form, Row, Window, Help) and a toolbar with icons for Select, Find, Add, Del..., Close, Seg..., New..., Run, Dis..., and Algo. Below the toolbar is a section for 'Object Type' with a dropdown menu. The main area contains a table with the following data:

	Name	Description	Ins Sys	Rep Sys	Object Type	Object Use	Related Object	Function Type
	B55BF302	Business Function Class - Chun	55	55	BSFN	1		
	FXS07182	Form for SAR 2323153	55	55	APPL			
	FXS07202	Form for SAR 2323153	55	55	APPL			
	R55731J	testing	55	55	UBE	161		
	RJK5506	Testing	55	55	UBE	161		
	RJK5515	Testing	55	55	UBE	161		
	15	test market renewal view	15	15	UBE	161		
	2100929	Test	55	55	APPL			
	37900B	Certificate of Analysis (COA) Extract	37	37	BSWW	340		
	40386	testing	55	55	UBE	161		
	40UI82B	workfile PID,FRNM,BSCD	45	45	BSWW	300		
	5500000	Test Table	55	55	TBLE	2		

Adding an Object

To add an object, you must indicate the object type, name it, and use the appropriate OneWorld development tool to define object specifications.

To add an object

On Work With Object Librarian

1. Complete the following field and click Add:
 - Object Type

You can choose from the following object types:

APPL	Interactive Application
BL	Business Function Library (use to set up your own custom library for custom DLLs)
BSFN	Business Function
BSVW	Business View
DSTR	Data Structure
GT	Media Object Data Structure
TBLE	Table Definition
UBE	Batch Application

The design form for the object type you are adding appears.

The Add Object form varies slightly depending on the object type you select.

Object Librarian - [Add Object]

File Edit Preferences Window Help

OK Cancel Dismiss Abort Links Display OLE Internet

Object Name: F00689 Object Type: TBLE

Description: Sales Transaction Update

Product Code: 42 Sales Management

Product System Code: 42 Sales Management Column Prefix:

Object Use: Undefined SAR Number:

2. On Add Object, complete the following required fields:

- Name
- Description
- System Code
- Category Code

For data structures only

The following fields are also required for specific object types:

- Column Prefix for Tables
- Src Long for business functions
- Dstr Type for data structures

3. Complete the following optional fields:

- Reporting System Code
- Object Use
- SAR Number

You can use processing options to make entry of a SAR Number mandatory. Refer to *Processing Options* for more information about creating processing options.

- Source Language

Source Language is used for business functions only.

The form example does not show the Source Language field because the example is for a table and not a business function. Source Language only applies when you add a business function.

4. Click OK.

Field	Explanation
Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
Ins Sys	A user defined code (98/SY) that identifies a J.D. Edwards system.
Rep Sys	A code that designates the system number for reporting and jargon purposes. See UDC 98/SY.
Object Use	Designates the use of the object. For example, the object may be used to create program, a master file, or a transaction journal. See UDC 98/FU.
Related Object	The Object Category is a functional grouping field. It is edited on the 98/OC UDC table.
Function Type	The Function Type groups business functions into Master, Major, and Minor categories based on the 98/E1 UDC table.
OCM Category	

Field	Explanation
ABC Priority	The ABC_Priority field indicates the status of an object as high, medium, or low. It is edited on the 98/E3 UDC table.
Process Type	The Process Type groups objects by operation, such as report, conversion, or batch process. It is edited on the 98/E4 UDC table.

Object Librarian consists of a Master table and a Detail table. The upper half of the form is the Master (header) information and includes information such as the Object Name, Description, Object Type, System Code, Reporting System Code, and Object Use.

Location	Path Code	Status Code	SAR	User	Date Modified	Time Modified
INTELNTC	APPL_PGF	1	2235840	TS5841423	6/9/98	10:07:30
INTELNTC	BLOBB733	1	2818498	KL5449350	2/19/99	11:24:31
JDESL	BLOBB733	1	2818498	KL5449350	2/17/99	13:52:40
INTELNTC	GLENNY	1	1684820	DM904413	4/13/98	16:33:20
INTELNTC	MSTB733	1	2235840	GA5807541	6/16/98	17:45:37
INTELNTD	PKGTEST	1	111	RT5741001	2/4/99	04:51:56
INTELNTC	PROD	1	111	RT5741001	2/4/99	04:51:56

The lower half of the form is the detail information and includes the following:

Location	Name of the server or workstations where the object resides
Path Code	Refers to the path code of the object. Used to keep track of objects and their location in OneWorld. For example, you might want a production path code where you make only quick fixes, and another development path code where you make long term enhancements. If two users have the same object checked out to their machines and they check them back into the same path code, the object checked in first will be overwritten by the one checked in last.

Status Code	Indicates the status of the object in the development cycle and whether it is located on the client or the server. Status Code “1” means the object is located on a server and is in production. Status Code “3” is used for object that are on a local workstation and in development.
SAR	An abbreviation for Software Action Request. This number is used by the J.D. Edwards’ Project Control system to keep track of all changes and activity pertaining to an object.
User	The identification code for a user profile.
Date Modified	This date indicates when the object was last modified either by an individual or by an upgrade or cumulative update.
Time Modified	Indicates the time of the last modification
Comments	Clients can use this field as they wish. J.D. Edwards typically uses this field for development comments between software releases.

Understanding Object Types and Naming Conventions

There are several standard objects types you can use to develop your applications.

Object naming conventions provide a methodology for identifying object file names used in interactive and batch applications. An interactive application consists of multiple objects, such as a table, business view, form, and event rules. Thus, before you can begin to create an application, you must add the objects required for your application.

J.D. Edwards recommends you use the following J. D. Edwards standards to name objects. These naming standards are current for this release.

See Also

- *Development Standards: Application Design Guide*

Tables

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

F = data table

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

00 - OneWorld 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

Provide up to a 60-character description for a table.

The table description is the topic of the table. If it came from the AS/400, it should be the same name as the file it represents, such as Address Book Master (F0101) and Item Master (F4101).

Business Views

The Object Librarian name for a business view can be a maximum of 10 characters and should be formatted as follows: **VzzzzzzzA**

V = business view

zzzzzzzz = should be the characters of the *primary* table

A = letter to designate which view. For example V0101A is the first view over the table F0101, V0101B is the second view over the same table.

External Developer Considerations

External development refers to creation of applications by developers outside of J.D. Edwards, such as Partners in Development and J.D. Edwards consultants that might create custom applications for specific clients. You must use caution when naming a business view to prevent collisions between J.D. Edwards and non-J.D. Edwards objects. When creating a new business view over a standard J.D. Edwards table, format the business view name as follows: **Vssss9999**, where

V = business view

ssss = the system code applicable to the enterprise

9999 = a unique next number or character pattern unique 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.



There should be one business view for each table that includes 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 in the case of a header/detail form. In this instance, two business views may be selected, one for the header portion of the form and one for the detail portion of the form.

Joined Views

Format the name for joined views using 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

Applications

The Object Librarian name for an application can be a maximum of 8 characters formatted as follows: **Pxxxxyyy**

The name field accepts up to 10 characters. However, if you enter more than 8 characters, the entry will be truncated.

P = application

xxxx = the system code

yyy = a next number, such as 001 and 002

Provide up to a 60-character description. It should reflect the subject of the forms within the application, such as Companies and Constants.

Forms

Forms Design automatically assigns a name using the following format: **WzzzzzzzzA**

W = form

zzzzzzzz = is the application name

A = the first form created in the application. It is usually, but not always, the entry point to the application; subsequent forms are assigned sequential letters, such as B for the second form, C for the third and so on.

For example, the application P0101 has two forms. The first form, Work with Addresses, is the entry point and is assigned the name, W0101**A**. The second form, Address Book Revisions, is assigned the name W0101**B**.

Provide a form description based on the form type:

Find/Browse - *Work With* followed by the subject of the application, such as Work With Companies, Work With Constants.

Fix/Inspect, Header Detail and Headerless Detail - should reflect the topic they cover, such as Supplier Information, Item Master Revisions, Purchase Order Entry.

For lower level forms, identify the form that called the form by appending the calling form's title, such as Enter Voucher - G/L Distribution.

When the title of a form includes a verb, use an active verb instead of a nominalization, such as Work With Vouchers.

Form Interconnection Data Structures

Form interconnection data structures are automatically created by the OneWorld tool set using the key fields in the business view.

You should change the data item name and description to describe the item being passed between forms.

Because Message forms do not have data structures, you must add at least the one member.

Business Function Event Rules (Named ER)

Object Type = BSFN

Source Language = NER

The source code for event rule business functions should be formatted as follows: **Nxxxxyyy**, such as N0400121.

N = event rule function

xxxx = the system code

yyy = a next number (the numbering assignments follow current procedures in the respective application groups.)

Business Functions

Object Type = BSFN

Source Language = C

The source code for business functions should be formatted as follows: **Bxxxxyyy**

B = business function

xxxx = the system code

yyy = a next number (the numbering assignments follow current procedures in the respective application groups.)



The next numbering for business functions and named event rules should not be shared to preserve the data structure or “D” names.

Business Function Data Structures

Object Type = BSFN

DSTR = 1

The data structure for business function event rules and business functions should be formatted as follows: **DxxxxyyyA**

D = data structure

xxxx = the system code

yyy = a next number (the numbering assignments follow current procedures in the respective application groups.)

A = If there are multiple data structures for a function, place an alphabetical character, such as A, B, C, and so on, at the end of the data structure name.

The data element in the data structure should use Hungarian Notation, with the data item alias appended. For example, if a data structure element is using LANO for the alias, the name would be mnSite_LANO.

Media Objects

Object Type = GT

The Object Librarian name for a media object can be a maximum of 8 characters and is formatted as follows: **GTxxxxyy**

GT = media object

xxxx = the file name excluding the letter F

For example, use GT0101 to name a media object for the F0101 file. If there is more than one media object for a file, then append an alphabetical character, such as A, B, C, and so on.

Provide up to a 60-character description. It should reflect the subject of the media object.

Batch Applications

Object Type = UBE

For batch applications, the name can be a maximum of 8 characters and should be formatted as follows: **Rxyyyyyy**, such as R09800, R30440.

R = batch (report) application

xx = system code

yyyyy = follow the same naming convention as on the AS/400 to fill in these last digits.

The Function Use field should follow the same naming standards as the AS/400, such as:

130-139 = Batch Processes

160-169 = Reports

Report Category Codes should follow the same standards as the Forms Design standards.

Modifying an Object

Before you can work with an existing object, you must check it out from the Object Librarian.

There can be more than one set of central objects. The central objects are usually stored on a server. For example, you might have a production path code where you make only quick fixes, and another development path code where you make long term enhancements. You must determine which set of central objects from which to check out.

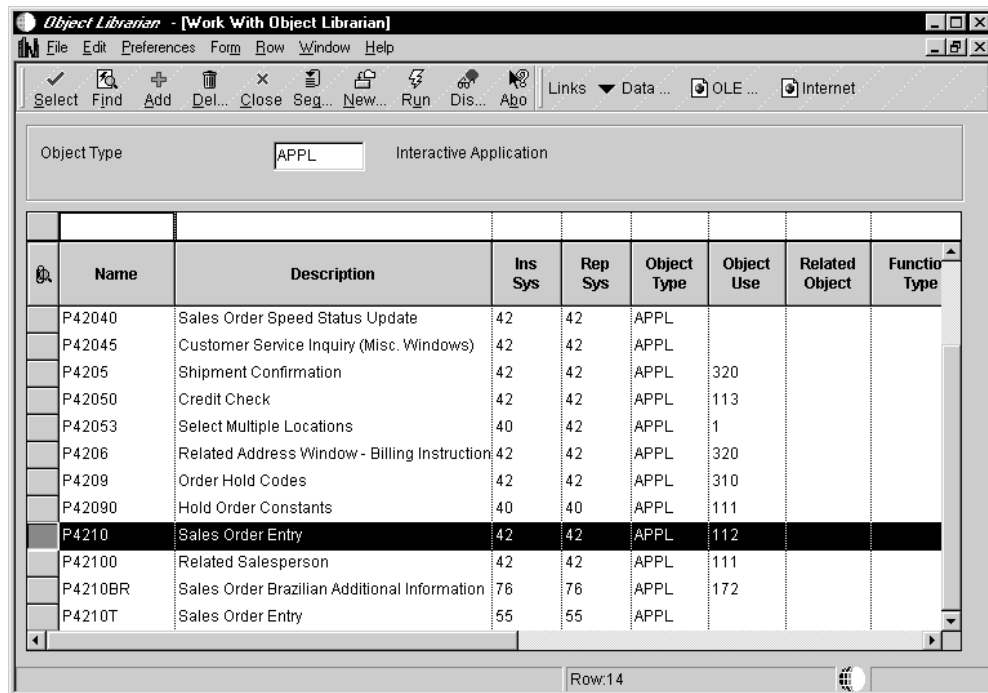
These sets of central objects are referred to as path codes. A developer checks objects out from a particular path code.

Check out the object for the appropriate environment. After completing your modifications, check in the object to the specific path code.

Checking in an object is a single database transaction. This ensures that an object will either be checked in completely or not at all. For example, if the power to the workstation was shut off during the check in of an object, the check in would be cancelled and the database specification files would revert back to their original state.

► To modify an object

On Work With Object Librarian



1. Choose an object and click Select.

Depending on the type of object you have chosen, the Object Librarian calls the appropriate Object Librarian Detail form. The grid displays status information about the object.

There is a processing option for for Object Librarian that you can enable to restrict multiple users from checking out objects.

Location	Path Code	Status Code	SAR	User	Date Modified	Time Modified
INTELNTC	APPL_PG	1	2953260	PV5557490	3/22/99	10:28:47
INTELNTC	MSTB733	1	2540781	DL5927235	10/21/98	18:35:48
INTELNTD	PKGTEST	1	2877316	GO5723697	3/1/99	14:49:08
INTELNTC	PROD	1	2938828	GO5723697	3/17/99	14:40:39
SAYLERB1	APPL_PG	3	2953260	BS5625464	3/22/99	11:40:36
BAIW22	APPL_PG	4	2183445	WB5886621	3/17/99	10:21:56
CHESLAKM	APPL_PG	4	111	MC5914223	2/26/99	16:45:56

You can prevent others from changing any column, except the modification comment column, by giving the line a status of 1. The following status codes are available:

- 1 = In Production
 - 2 = In Test
 - 3 = In Development
 - 4 = Custom Version
 - 5 = In Use by Vocabulary Overrides
2. Select the grid row that reflects the path code and server from which you want to check out the object.

You cannot copy object specification records from one person's workstation to another. All objects must go through a server. An object on one workstation must be checked into the server in order for someone to check the object out to another workstation.

3. From the Row menu, choose Check Out.

Check Out copies and converts the object specifications to your workstation. For large objects this process may take a little while.

4. From the Form menu, choose Design to access the selected object's design tool.
5. Make your modifications.

You should test your modifications to make sure they work before you check the object back in.

6. From the Row menu, choose Check In to copy the modified object to the server.

Object Librarian Check In displays available path codes to which you can check in your object. Just as you can check out an object from various path codes, you can choose the path code to which you want to check in.

Path Code	Location	Description
APPL_81	INTELNTC	B81 PATH CODE
APPL_PG	INTELNTC	Applications Development
B7312	INTELNTC	MASTERS B73.1.2
BLDCTL	BUILD21	Build 21 control table
BLOBB732	INTELNTC	oneworld rdb blob
BLOBJSH	INTELNTC	oneworld rdb blob
CANDACETST	INTELNTC	Test for Candace
FDA_RDA	RAIZEND	FDA and RDA specification path
GLENNY	YINGERG	Do not Use : Test for Yinger

7. Select the grid row that represents the path code and server to which you want to check in.

You should coordinate your work with other users to avoid unintentionally overwriting their work when the object is checked in. Any user with proper security can check out and modify an object. The last object that is checked in overwrites any previous copies.

The Object Librarian does not store backed up copies of objects.

8. To remove the check out record and keep the modified specifications on your workstation, choose Erase Check Out from the Form menu. This keeps the

Object Librarian Detail table cleaned up. If you do not intend to check in the modifications, then erase the check out.

9. To delete modified specifications, click Delete.

See Also

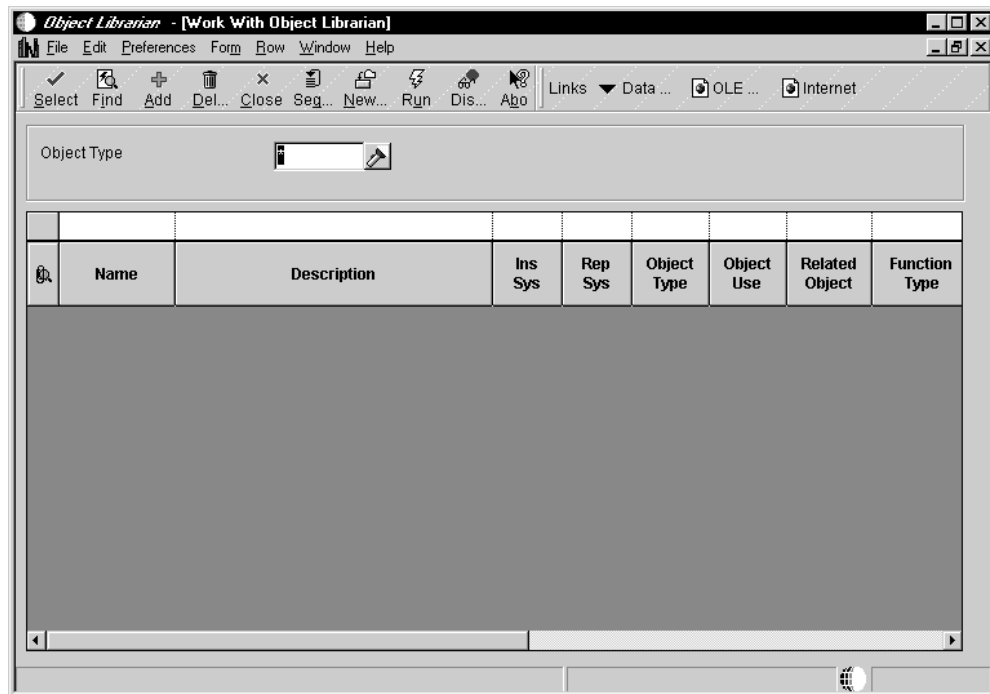
- *Appendix A: OneWorld Modification Rules*

Deleting an Object

If you delete an application from Object Librarian, all of the application's associated forms are also deleted from the F9865 table.

► To delete an object

On Work With Object Librarian



1. Choose an object and click Delete.

Deletions of objects that are only on a workstation do not require a password. Deletions of objects checked into the Object Librarian server are secured by a hardcoded password. In this case you must also complete the following steps.

2. Enter the password.



You may need to call the J.D. Edwards Response Line to get this password.

3. After you enter the password, you must complete the Delete Object Specifications form.

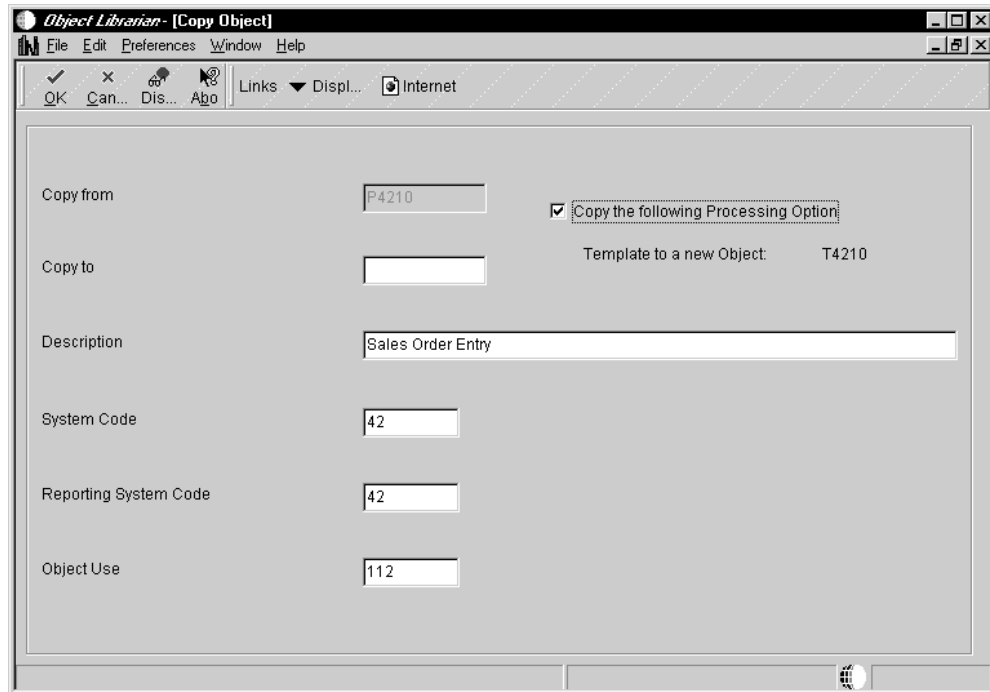
4. Confirm deletion of the object.

Copying an Object

You can use Object Librarian to copy objects. If the object you are copying is an interactive or batch application with a processing option template attached, you can choose to copy the template with the application. Object Librarian will automatically update any template references in your application from your old template to the new one. You can also copy data structure objects (DSTR), including processing option templates.

► To copy an object

On Work With Object Librarian



1. Choose the object you wish to copy.
2. From the File menu, choose Copy.
3. Complete the following field:
 - Copy To
4. If you want to copy a processing option template that is attached to your application or report, turn on the following option:
 - Copy the following Processing Option

Working with Object Librarian Codes

Use Object Librarian Codes to customize software builds and installations. For example, you can group objects by category to identify only those objects you want to rebuild, rather than rebuilding the entire system. Or you can assign install/merge codes to determine whether user modifications are overwritten by or merged with an install version.

► To work with Object Librarian Codes

On Work With Object Librarian

1. Choose the object for which you want to assign object codes.
2. From the Row menu, choose Object Codes.

3. Click either of the following tabs, depending on the desired task:
 - Category Codes
 - Install / Merge Codes
4. To group objects by category codes, click the Category Codes tab and complete the following fields:
 - Object Use
 - Related Object
 - Programming Code (ABC)
 - CNC Code (ABC)
 - Parent Object
 - Business Function Type
 - Event Rule Replaceable
 - Process Type
 - Structure Type

The fields that appear will vary depending on the type of object you choose.

Object Librarian - [Object Librarian Codes]

File Edit Preferences Window Help

OK Cancel Dismiss Abort Links Display Internet

Object Name: P4210 Sales Order Entry

Object Type: APPL Interactive Application

Category Codes Install/Merge Codes

Object Use: 112 Transaction Proc.

Related Object: [Empty] [Browse]

Programming Code (ABC): A High Impact

CNC Code (ABC): [Empty]

Parent Object: [Empty]

5. To assign merge/install codes, click the Install/Merge Codes tab and complete the following fields:
 - System Code
 - Reporting System Code
 - Omit Option

- Optional File
- Common File
- Copy Data (Y/N)
- Global Build Option

Object Librarian - [Object Librarian Codes]

File Edit Preferences Window Help

OK Cancel Dismiss Abort Links Display Internet

Object Name: P4210 Sales Order Entry

Object Type: APPL Interactive Application

Category Codes Install/Merge Codes

System Code	42	Sales Management
Reporting System Code	42	Sales Management
Optional File	N	Not an Optional File
Common File	N	No
Copy Data (Y/N)	N	
Global Build Option		Not Used in JDEGLOBAL

Field	Explanation
Related Object	The Object Category is a functional grouping field. It is edited on the 98/OC UDC table.
Common File	A file designated as a Common Library Data File is copied into the users common library when the Create User Production Library job is run.
Copy Data (Y/N)	Used to indicate if a file and its data are copied into production. A value of N moves the file without data during a file copy. When creating a production data library from JDFDATA, this field designates whether the data is included in the copy.
Category Codes	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.
Install / Merge Codes	Used for software builds and installations.
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.

Field	Explanation
Object Use	Designates the use of the object. For example, the object may be used to create program, a master file, or a transaction journal. See UDC 98/FU.
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Business Function Type	The Function Type groups business functions into Master, Major, and Minor categories based on the 98/E1 UDC table.
Event Rule Replaceable	The ER Replace field indicates whether or not a business function can be replaced by Event Rules. It is edited on the 98/E2 UDC table.
Programming Code (ABC)	The ABC_Priority field indicates the status of an object as high, medium, or low. It is edited on the 98/E3 UDC table.
Process Type	The Process Type groups objects by operation, such as report, conversion, or batch process. It is edited on the 98/E4 UDC table.
CNC Code (ABC)	Object Librarian Category Code 05 is not in use at this time.
Structure Type	The Data Structure Type indicates the type of structure as follows: 1 Function Data Structure 2 Processing Option Template It is edited on the 98/E6 UDC table.
Omit Option	Designates records in Software Versions Repository file that are not included on new releases. These codes are as follows: - H Held from all releases - X Omit from all releases - S Omit Source from all releases - O Omit Execution Object from all releases

Field	Explanation
Optional File	Valid codes are: Y Designates a file as an Optional Data File if there are some situations where the file may not be needed at a client installation. The explanation of these situations can be found in the Generic Rate/Message information for that file for Generic Rate/Message Type 96/OF. All of these files that exist in a specified library can be listed in the Optional File Report on menu A9645. O Designates that the file is designated for omission. Examples are compile files or special files like JDE User Profiles file.
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.
Reporting System Code	A code that designates the system number for reporting and jargon purposes. See UDC 98/SY.

See Also

- See the *Package Management Guide* for more information about building and installing packages.

Displaying a Reference List of Business Functions

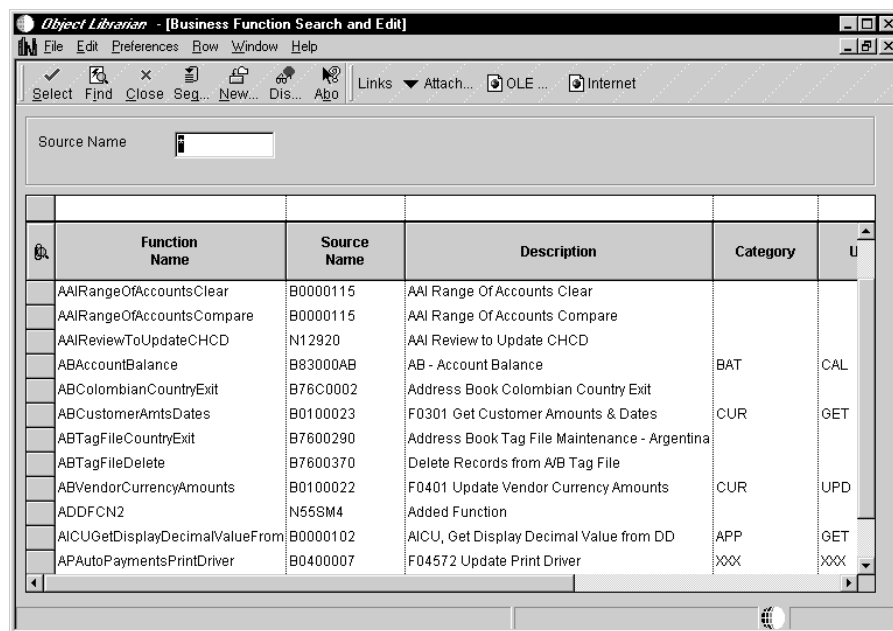
The Business Function Search and Edit form displays a list of business functions. This list is especially helpful when a business function includes source for two separate business functions and the module is named differently than the business functions it contains. For example, the description for business function B9200005 is “Data Dictionary - Glossary Item Functions,” but it includes source for the “Update Related Glossary Tables” and “Retrieve Error Message Info” business functions.

The Business Function Search and Edit form is also the entry point to the Business Function Source Librarian, where you can design and build business functions.

► To display a reference list of business functions

On Work with Object Librarian

1. From the Form menu, choose Function Search.



2. To access the Business Function Source Librarian where you can edit a business function, choose the desired business function from the grid.

Field	Explanation
Category	The general category of the type of data processed by the business function.
Use	The primary function or process that occurs during the business function.
Template Name	The character name of the data structure template.
Source Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Function Name	The Function Name is a valid “C” function name.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.

Working with the Business Function Source Librarian

Use the Business Function Source Librarian to:

- Design a business function
- Build a business function



To design a business function

On Business Function Source Librarian

Location	Path Code	Status Code	SAR	User	Date Modified	Time Modified
INTELNTC	APPL_PG	1	2820467	ND5644286	1/22/99	13:19:59
INTELNTC	MSTB733	1		JDE	12/15/97	08:00:00
INTELNTD	PKGTEST	1	2820467	CB5930119	2/5/99	14:02:31
INTELNTC	PROD	1	2820467	CB5930119	2/5/99	14:02:31
PANSUVANC	APPL_PG	4	2820467	CP5912148	2/5/99	13:44:48
PRYORG	PROD	4	2820467	MR5868124	2/5/99	11:18:45

1. From the Form menu, choose Design.

Function Name	Description	Function Category	F
AAIRangeOfAccountsClear	AAI Range Of Accounts Clear		
AAIRangeOfAccountsCompare	AAI Range Of Accounts Compare		
F0012GetAAIRangeOfAccounts	F0012 Get AAI Range Of Accounts		GET

2. On Business Function Design, you can define the business function.

Once you add a function, the function name cannot be changed. The name must also be unique across the enterprise.

Field	Explanation
Source Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Programming Language	The source language code identifies the programming language that a business function is written in.
Description	A free-form, 80-character text field.
Parent DLL	DLL into which the Object is built.
Business Function Location (C/S)	The path from where the business function is retrieved.

You can add the Parent DLL and business function location using Business Function Design. See *Business Functions* in this guide for more information.

The default parent DLL for all business functions is CALLBSFN. You can specify that certain business functions are only meant to be built for workstations and not for servers. This way in a distributed environment, you can specify which business functions are distributed and which ones are not.

Working with Promotion Manager

You can use Promotion Manager to see which objects someone has checked out. You can also track who has objects checked out to local workstations. Promotion Manager allows you to search for objects based on specific requirements including, object name, SAR, User ID, status, or Workstation Name. Promotion Manager is real-time; it is not a log.

Promotion Manager is useful if you will be out for several days, for example on vacation. You should check your objects in as needed before you leave.

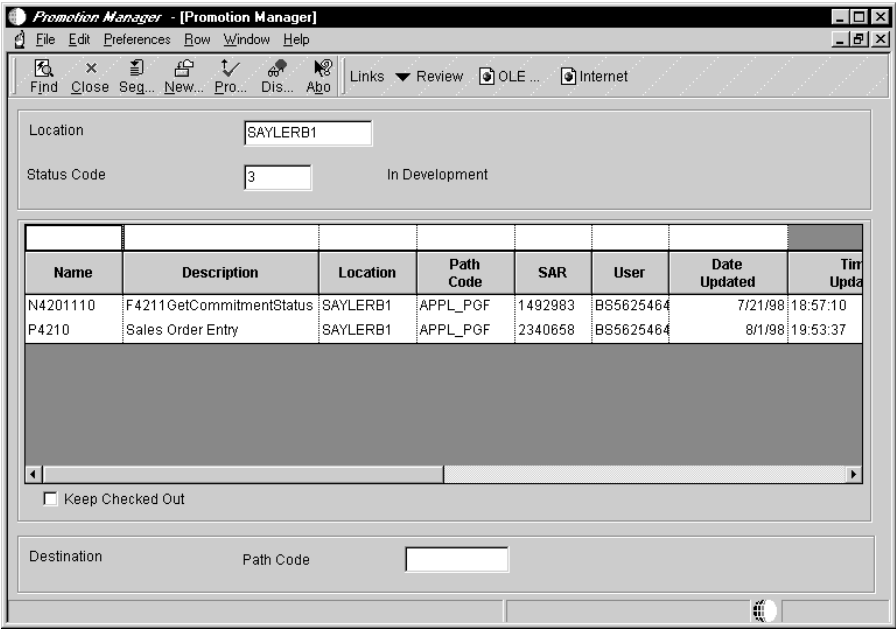
You can check in a series of objects at the same time. This is useful so you can check in only those objects connected to a certain Software Action Request (SAR) number or to move objects from development environments to testing environments. Promotion Manager also allows you to access an object's detail information in Object Librarian so you can edit the object's merge code.

Using Promotion Manager

To use Promotion Manager

On Object Management (GH9081)

1. Double-click on Promotion Manager (P98603)



2. Choose one of the following options from the row menu:

- Review

You can choose an item in the detail area on the grid , then choose Review. Review takes you directly to Object Librarian. You can then perform other tasks such as go to into design, check the item in, or erase the check out
- Promote

You can use promote to check an object into the path code you specify in the Path Code.
- Merge

You can use merge to specify whether an object will be merged in with a OneWorld object. You indicate the path code and all objects you check into that path code will have the same merge option.

Field	Explanation
Location	For World, the Location indicates the machine (server or client). For OneWorld, the Location or Machine Key indicates the name of the machine on the network (server or workstation).
Status Code	This code determines the status of the software in the development cycle.

Field	Explanation
Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
Path Code	The Path Code is a pointer to a set of OneWorld objects, and is used to keep track of sets of objects and their locations within OneWorld.
SAR	An abbreviation for Software Action Request (SAR). This number is used by the J.D. Edwards project control system to keep track of all charges and activity by individual SAR number.
User	For World, the IBM-defined user profile. For OneWorld, the identification code for a user profile.
Date Updated	The date of the last update to the file record.
Time Updated	The time the object was last checked in.

Data Dictionary

Just as a dictionary contains word definitions, the J.D. Edwards data dictionary is a central repository that contains data item definitions and attributes. 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. The data item attributes determine how a data item:

- Appears on reports and forms (such as number of decimals, and default values)
- 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 dynamic. That is, any changes made to a data item are effective immediately across all applications. 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.

Use the data dictionary to create, view, and update attributes for data items. You can copy a data item that has similar attributes and modify it for your specific needs, rather than creating a new one from scratch. This may be quicker and easier than creating a new one. If you do this, you must modify the alias to distinguish between the copy and the original.

When you change a data item the changes are immediately reflected throughout the OneWorld tools at runtime. Changing the type and any of the attributes may affect how your data is stored and cause discrepancies between records.



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, the application will fail.

This section describes:

- ☐ Using the data dictionary
- ☐ Defining a data item

Before You Begin

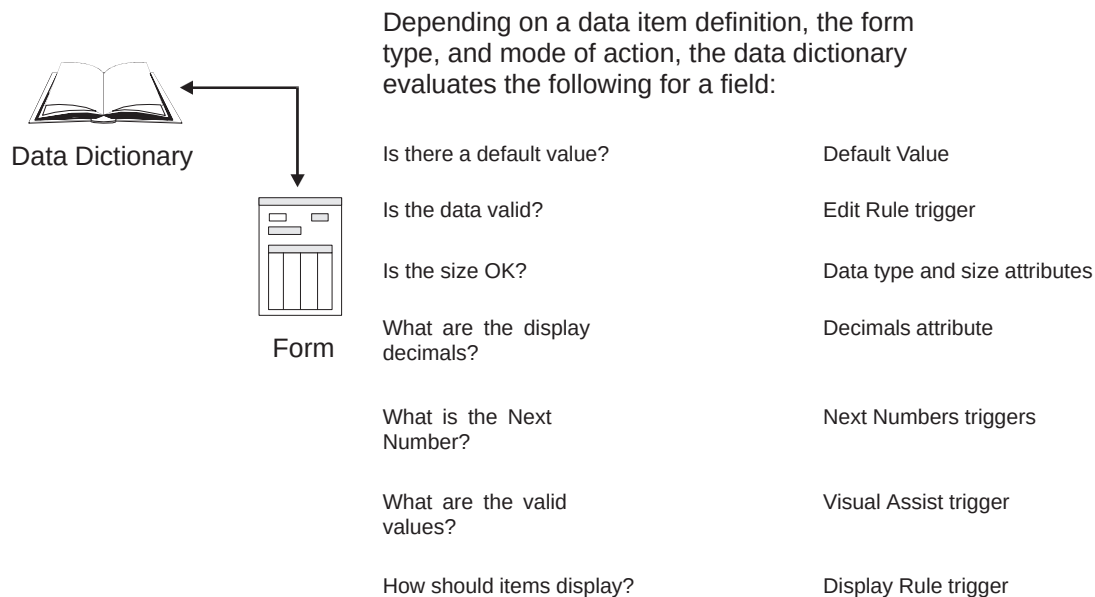
Before you begin working with the data dictionary, you should be familiar with the concepts in:



☐ *OneWorld Foundation Guide*

☐ *Object Librarian*

How the Data Dictionary is Used at Runtime



At runtime, applications access the following data dictionary 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 by an asterisk (*) can be overridden in Forms Design and Report Design. In these instances, the application retrieves the overrides, if any exist.

Naming Data Items

The following illustrates the naming components for a data item:

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

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

See Also

Development Standards: Application Design Guide for naming standards.

Storing the Data Dictionary and Data Dictionary Items

The data dictionary is stored in two places:

- The central object data dictionary is stored in a relational database. All changes to the data dictionary that will be replicated a 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 necessary for the applications that you are using. This replication occurs when you use an application for the first time after installing OneWorld. This data dictionary information is stored on your workstation in a permanent cache under the same local path code/spec directory as the following global tables:

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

If data items are changed and you want the changes replicated 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 a machine that is set up as a subscriber signs onto OneWorld, the permanent cache for that data item will be deleted. Then the next time they use an application that has that changed data item, OneWorld will detect that the information is not in the permanent cache and retrieve the information from the publisher data dictionary (the relational database tables).

If you are using OneWorld and World coexistence, you must maintain two data dictionaries. You cannot share one dictionary because in World 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 OneWorld. Some of the file formats in World and OneWorld are also different.

See Also

- Data Dictionary Administration in the *System Administration Guide*.

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 OneWorld are stored as data items. Use the data dictionary glossary item application to display error messages because you do not need to use all the fields required for regular data items.

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 leverage your time and code because you only have to create the business logic once, but you can use it within multiple applications. Default triggers ensure accuracy and integrity of data across all applications.

You use triggers to:

- 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 tabs into a field. The types of visual assists are:
 - calendar
 - calculator
 - Search and Select of another file
 - UDC
- Establish rules and procedures that are embedded in the editing and formatting of a field's data

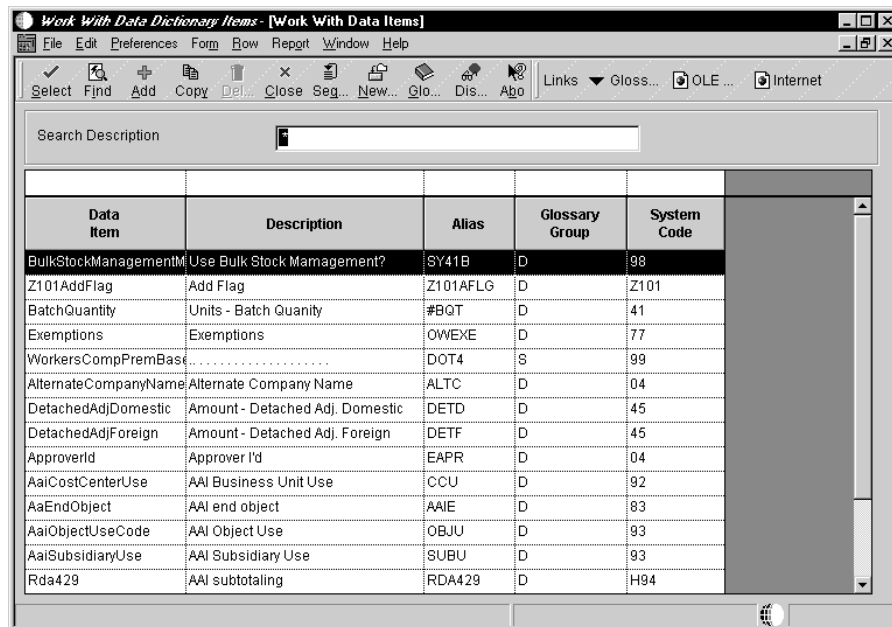
-
- Determine a next number scheme that should be used when assigning a number to data

Using the Data Dictionary

The Work With Data Items form is the entry point for data dictionary items.

► To use the Data Dictionary

From the Data Dictionary menu (GH951), choose Work With Data Items (P92001).



The screenshot shows a software window titled "Work With Data Dictionary Items - [Work With Data Items]". It features a menu bar (File, Edit, Preferences, Form, Row, Report, Window, Help) and a toolbar with icons for Select, Find, Add, Copy, Del, Close, Seg, New, Glo, Dis, and Ago. Below the toolbar is a search bar labeled "Search Description". The main area contains a table with the following data:

Data Item	Description	Alias	Glossary Group	System Code
BulkStockManagement	Use Bulk Stock Management?	SY41B	D	98
Z101AddFlag	Add Flag	Z101AFLG	D	Z101
BatchQuantity	Units - Batch Quantity	#BQT	D	41
Exemptions	Exemptions	OWEXE	D	77
WorkersCompPremBase	Workers Comp Prem Base	DOT4	S	99
AlternateCompanyName	Alternate Company Name	ALTC	D	04
DetachedAdjDomestic	Amount - Detached Adj. Domestic	DETD	D	45
DetachedAdjForeign	Amount - Detached Adj. Foreign	DETF	D	45
ApproverId	Approver Id	EAPR	D	04
AaiCostCenterUse	AAI Business Unit Use	CCU	D	92
AaEndObject	AAI end object	AAIE	D	83
AaiObjectUseCode	AAI Object Use	OBJU	D	93
AaiSubsidiaryUse	AAI Subsidiary Use	SUBU	D	93
Rda429	AAI subtotalling	RDA429	D	H94

Field	Explanation
Alpha Description	Categorizes data item names. Enter text in upper and lower case. The system uses this field to search for similar data items. To enter an alpha description, follow these conventions: - Dates – Begin all Date fields with Date - Amounts – Begin all Amount fields with Amount - Units – Begin all Unit, Quantity, and Volume fields with Units - Name – Begin all 30-byte description fields with Name - Prompt – Begin any Y/N prompting field with Prompt - Address Number – Begin all address numbers (employee, customer, owner) with Address Number
Search Description	Categorizes data item names. Enter text in upper and lower case. The system uses this field to search for similar data items. To enter an alpha description, follow these conventions: - Dates – Begin all Date fields with Date - Amounts – Begin all Amount fields with Amount - Units – Begin all Unit, Quantity, and Volume fields with Units - Name – Begin all 30-byte description fields with Name - Prompt – Begin any Y/N prompting field with Prompt - Address Number – Begin all address numbers (employee, customer, owner) with Address Number
Alias	For World, the RPG data name. This data field has been set up as a 10-byte field for future use. Currently, it is restricted to 4 bytes so that, when preceded by a 2-byte table prefix, the RPG data name will not exceed 6 bytes. Within the Data Dictionary, all data items are referenced by this 4-byte data name. As they are used in database tables, a 2-character prefix is added to create unique data names in each table specification (DDS). If you are adding an error message, this field must be left blank. The system assigns the error message number using next numbers. The name appears on a successful add. You should assign error message numbers greater than 5000. Special characters are not allowed as part of the data item name, with the exception of #, @, \$. You can create protected data names by using \$xxx and @xxx, where you define xxx. For OneWorld, a code that identifies and defines a unit of information. It is an 8-character, alphabetical code that does not allow blanks or special characters such as: % & , . +. Create new data items using system codes 55-59. The alias cannot be changed.

Field	Explanation
G G	For World, a code which designates a type of data used to select data dictionary terms for printing. See User Defined Codes, system code '98', record type 'GG'. The data item names for error messages are assigned automatically. NOTE: If you need to assign your own error message numbers, use 4 digit numbers greater than '5000'. The data item name for a non-database field (used on a video or report but not in a file – glossary group U) must begin with a #, \$ or @. For help text (glossary group H), the data dictionary “Inquiry/Revision Program” field may be used to specify the name of a follow-on item. To create your own messages for the IBM message file (glossary group J), begin the data item name with your own three characters (e.g., CLT0001). For OneWorld, validates against UDC H98/DI. A code used to designate the type of data item. Items in glossary group D or S can be included in database tables. Items in other glossary groups (for example, error messages) cannot be added to a table.
Data Item	An identifier that refers to and defines a unit of information. It is a 32-character, alphabetical field that does not allow blanks or special characters such as % & , . +. The data item cannot be changed. It forms the C-code data name (for example AddressNumber) that is used in business functions, data structures, and event rules. Also identify a data item by the alias or alpha description.
Syst Code	A user defined code (98/SY) that identifies a J.D. Edwards system.

Defining a Data Item

You define data item specifications when you add, modify, or copy a database data item. Defining a data item includes the following tasks:

- ☐ Naming a data item
- ☐ Defining general information
- ☐ Attaching default triggers
- ☐ Updating the glossary

You can also perform one or more of the following tasks:

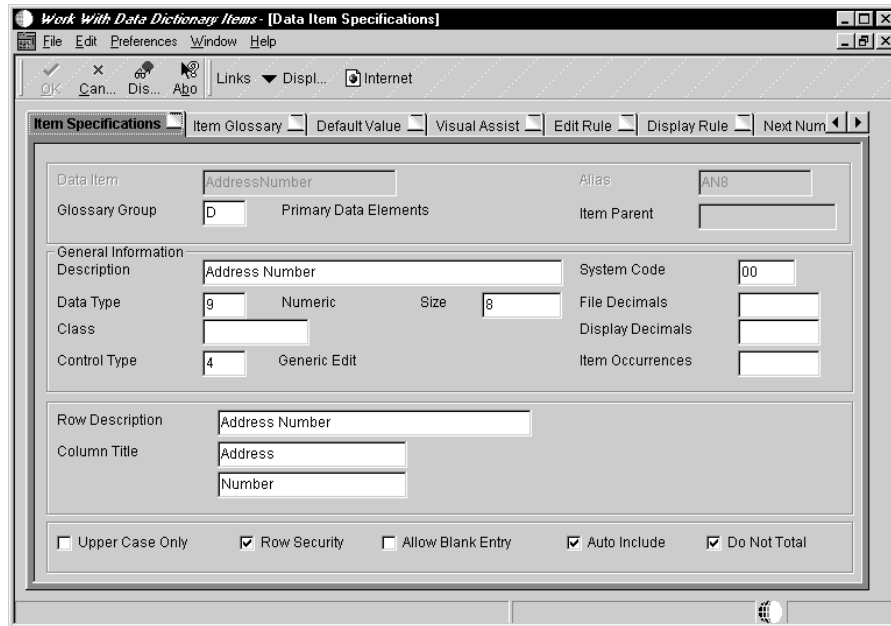
- ☐ Attaching a default value trigger
- ☐ Attaching a visual assist trigger
- ☐ Attaching an edit rule trigger
- ☐ Attaching a display rule trigger
- ☐ Attaching a user defined code trigger
- ☐ Attaching a next number trigger
- ☐ Attaching a smart field trigger

Naming a Data Item

You must assign a unique name and attributes to a new data dictionary data item before it can be used by an application.

► **To name a data item**

1. On Work with Data Items click Add.



2. On Data Item Specifications, complete the following fields:
 - Data Item
 - Alias
 - Glossary Group

Data Dictionary Item Naming Conventions

J.D. Edwards recommends the following naming standards. Refer to the *Development Standards: Application Design Guide* for more information about naming standards.

Data Item Name

The data item name is a 40-character, alphabetical field that identifies and defines a data item.

The data item name forms the C-code data name (for example AddressNumber) that is used in business functions, data structures, and event rules.

When creating a J.D. Edwards data dictionary item, do not use a Y or Z in the first character of the data item name. Y and Z are reserved for Partners in Development business partners. (J.D. Edwards data items beginning with Y or Z may exist prior the B73.2 release of OneWorld. These items will remain J.D. Edwards items.)

Blanks and the characters % & , . + are not allowed as part of the data item name in OneWorld.

You can also identify a data item by the alias or alpha description.

Data Item Name for an External Data Dictionary Item

When creating an external data item, you must use a Y or Z in the first character of the data item name to distinguish an external data dictionary item from a J.D. Edwards data dictionary item.

The data item name can be a maximum of 40 alphanumeric characters and uses the following format:

Yssssdddddddddddddddddddddddddddddddd, where:

Y or Z = designates an external data dictionary item and is the first digit of any J.D. Edwards-assigned external system code

ssss = the system code number assigned by the partners in development system administrator, or 55xx-59xx for enterprise-level development of new modules, or 60xx-69xx for J.D. Edwards custom development

dddddddddddddddddddddddddddddddd = the name of the data item name

Data Item Alias

The data item alias is an 10-character alpha code. If the data dictionary item is exclusive to OneWorld applications, the alias is five or more characters in length. When adding a data item that will be used in a table by an RPG program, the alias must not exceed four characters.

The data item alias is used when searching, within database routines (application program interfaces used in business functions), and within Table Design when creating a table. For each table, a prefix is added to the alias, which makes it unique to that table. For example, the alias ABMCU indicates that MCU is within the Address Book.

Blanks and the characters % & , . + are not allowed as part of the data item alias in OneWorld.

You can also identify a data item by the data item name or alpha description.

Once added, you cannot change the data item or alias.

Alias for an External Data Dictionary Item

An external data dictionary item is one that is created by a developer outside of J.D. Edwards for use in OneWorld. For external data items, the data dictionary alias can be a maximum of 10 alphanumeric characters and uses the following format:

Yssssddd, where:

Y or Z = designates external data dictionary item and is the first digit of any J.D. Edwards-assign external system code.

ssss = the system code number assigned by the partners in development system administrator, or 55xx-59xx for enterprise-level development of new modules, or 60xx-69xx for J.D. Edwards custom development

ddd = the name of the data dictionary item

Field	Explanation
Data Item	An identifier that refers to and defines a unit of information. It is a 32-character, alphabetical field that does not allow blanks or special characters such as % & , . +. The data item cannot be changed. It forms the C-code data name (for example AddressNumber) that is used in business functions, data structures, and event rules. Also identify a data item by the alias or alpha description.

Field	Explanation
Alias	For World, the RPG data name. This data field has been set up as a 10-byte field for future use. Currently, it is restricted to 4 bytes so that, when preceded by a 2-byte table prefix, the RPG data name will not exceed 6 bytes. Within the Data Dictionary, all data items are referenced by this 4-byte data name. As they are used in database tables, a 2-character prefix is added to create unique data names in each table specification (DDS). If you are adding an error message, this field must be left blank. The system assigns the error message number using next numbers. The name appears on a successful add. You should assign error message numbers greater than 5000. Special characters are not allowed as part of the data item name, with the exception of #, @, \$. You can create protected data names by using \$xxx and @xxx, where you define xxx. For OneWorld, a code that identifies and defines a unit of information. It is an 8-character, alphabetical code that does not allow blanks or special characters such as: % & , . +. Create new data items using system codes 55-59. The alias cannot be changed.
Glossary Group	For World, a code which designates a type of data used to select data dictionary terms for printing. See User Defined Codes, system code '98', record type 'GG'. The data item names for error messages are assigned automatically. NOTE: If you need to assign your own error message numbers, use 4 digit numbers greater than '5000'. The data item name for a non-database field (used on a video or report but not in a file - glossary group U) must begin with a #, \$ or @. For help text (glossary group H), the data dictionary "Inquiry/Revision Program" field may be used to specify the name of a follow-on item. To create your own messages for the IBM message file (glossary group J), begin the data item name with your own three characters (e.g., CLT0001). For OneWorld, validates against UDC H98/DI. A code used to designate the type of data item. Items in glossary group D or S can be included in database tables. Items in other glossary groups (for example, error messages) cannot be added to a table.

Defining General Information

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

The screenshot shows a software window titled "Work With Data Dictionary Items - [Data Item Specifications]". It has a menu bar (File, Edit, Preferences, Window, Help) and a toolbar with buttons like OK, Cancel, Dis..., and a mouse cursor icon. Below the toolbar are several tabs: "Item Specifications" (selected), "Item Glossary", "Default Value", "Visual Assist", "Edit Rule", "Display Rule", and "Next Num". The "Item Specifications" tab contains several input fields and checkboxes. The "Data Item" field is "AddressNumber" and the "Alias" field is "AN8". The "Glossary Group" is "D" and "Primary Data Elements" is empty. The "Item Parent" field is empty. The "General Information" section includes: "Description" (Address Number), "System Code" (00), "Data Type" (9 Numeric), "Size" (8), "File Decimals" (empty), "Display Decimals" (empty), "Item Occurrences" (empty), "Class" (empty), and "Control Type" (4 Generic Edit). The "Row Description" field is "Address Number". The "Column Title" field has two entries: "Address" and "Number". At the bottom, there are checkboxes: "Upper Case Only" (unchecked), "Row Security" (checked), "Allow Blank Entry" (unchecked), "Auto Include" (checked), and "Do Not Total" (checked).

► To define general information

On Data Item Specifications

1. 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

Item occurrences is used for arrays. This allows you to create an item as a child of another item. The data dictionary performs validation to ensure that attributes are consistent between the parent and the child. If you change the parent item, the changes are duplicated to the child items. The data item names use the parent data item name and a number, for example, 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

Field	Explanation
Description	Categorizes data item names. Enter text in upper and lower case. The system uses this field to search for similar data items. To enter an alpha description, follow these conventions: - Dates - Begin all Date fields with Date - Amounts - Begin all Amount fields with Amount - Units - Begin all Unit, Quantity, and Volume fields with Units - Name - Begin all 30-byte description fields with Name - Prompt - Begin any Y/N prompting field with Prompt - Address Number - Begin all address numbers (employee, customer, owner) with Address Number
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.

Field	Explanation
Data Type	The style or classification of data, such as numeric, alphabetic, and date. J.D. Edwards recommends that you do not change the data item type if it is used within an existing application. Otherwise, you must regenerate your table and review all business functions that use this data item. Data types include: - Character - A single letter, always the size of 1 - Date - Date - Integer - Integer - A blob is a huge chunk of information.formatted with a delimiter. It is not typically seen in applications. - Character (Blob) - can be translated from EBCDIC (8-bit character code widely used on IBM mainframes) to ASCII (7-bit code) - Binary (Blob) - cannot be translated, appears in machine code, and is found as an executable file under win.help - Binary - Based on the number system 2. Usually a combination of the digits 0 and 1. Can also be used to represent on and off or true and false. - String - always the same size or length - Variable String - an item of variable size - Identifier (ID) - used under the covers for controls. Used to write a C program and reference third party software that returns a pointer. A J.D. Edwards API then save the pointer that references the ID. The parameter passed to the C program is the ID. - Numeric - long integer
Size	The field size of the data item. NOTE: All amount fields should be entered as 15 bytes, 0 decimals, and the data item type should be P (packed).
File Decimals	The number of positions to the right of the decimal of the data item that are stored. Addresses how the information is stored in the database.
Display Decimals	Designates the number of decimals in the currency, amount, or quantity fields the system displays. For example, U.S. Dollars would be 2 decimals, Japanese Yen would be no decimals, and Cameroon Francs would be 3 decimals.

Field	Explanation
Control Type	Defines the type of graphical user control that is associated with the data item. For example, a data item can appear as a push button, check box, user defined code, and so on. Control type is used by Forms Design to automatically add the correct control to a form for a specific data item. For example, if a data item will normally be used as a check box, then the Data Dictionary control type should be a check box. When you use a Quick Form, the data item will default as a check box control instead of a generic edit on your form. You can override in Forms Design; however, you should anticipate how it will most commonly be used and set it up accordingly.
Row Description	For World, creates the title on text and reports. It is used in a manner similar to the column description in the query facility. It should be less than 35 characters. Use abbreviations whenever possible. For example: - U/M Units of measure - YTD Year-to-date - MTD Month-to-date - PYE Prior year end - QTY Quantity - G/L General ledger - A/P Accounts payable - DEPR Depreciation For OneWorld, the row description is used for field description on forms and reports.
Column Title	The first line of description that will be used in column headings on a report or form. This description should be no larger than the data item size, if possible. If the column heading is only one line, it should be placed in this column. Use the second line of the Column Title when one is not clear.
Class	Data item class. A class defines the essential attributes and characteristics of a data item. Informational only.
Item Occurrences	In setting up a data item in the data dictionary, you may specify a number of array elements. This will cause the automatic creation of one additional data item for each array element. The array data item names are restricted to certain lengths depending on the number of array elements: 3 bytes - 1 to 9 elements 2 bytes - 10 to 99 elements 1 byte - 100 to 999 elements
Upper Case Only	If the value of this field is a Y, the user will be allowed to enter only upper case characters in the entry control.

Field	Explanation
Allow Blank Entry	Turning this option on allows a blank value to be written to the database under the following conditions: (1) If the field is edited against a UDC table, a blank value will be allowed regardless of whether a blank value is valid for the table. (2) If the field is specified to be a mandatory entry, a blank value will be allowed as a valid entry.
Row Security	A flag to indicate that the field can be used in setting up row security.
Auto Include	Determines whether this column should be automatically included in all database fetches to tables that contain this item. This option should only be set for items that are essential for certain database trigger processes or security validation.

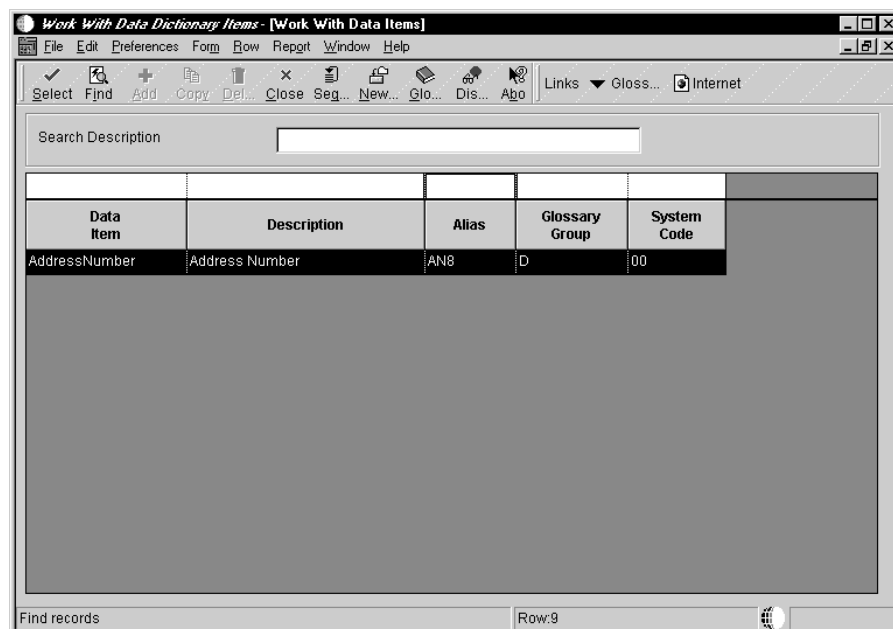
Updating Data Items for Languages

If you add a new data item, remember that the row and column are for the base language only, unless you update the description for another language.

► To update a data item for languages

On Work with Data Items

1. Find the item you wish to update.



2. From the Row menu, choose Descrip. Overrides

Language	Jargon Code	Row Description	Column Title 1	Column Title 2
		Address Number	Address	Number
	41	Supplier	Supplier	Number
	42	Sold To	Sold To	Number
	43	Supplier	Supplier	Number
	44	Supplier	Supplier	Number
	46	Supplier	Address	Number
	48	Customer Number	Address	Number

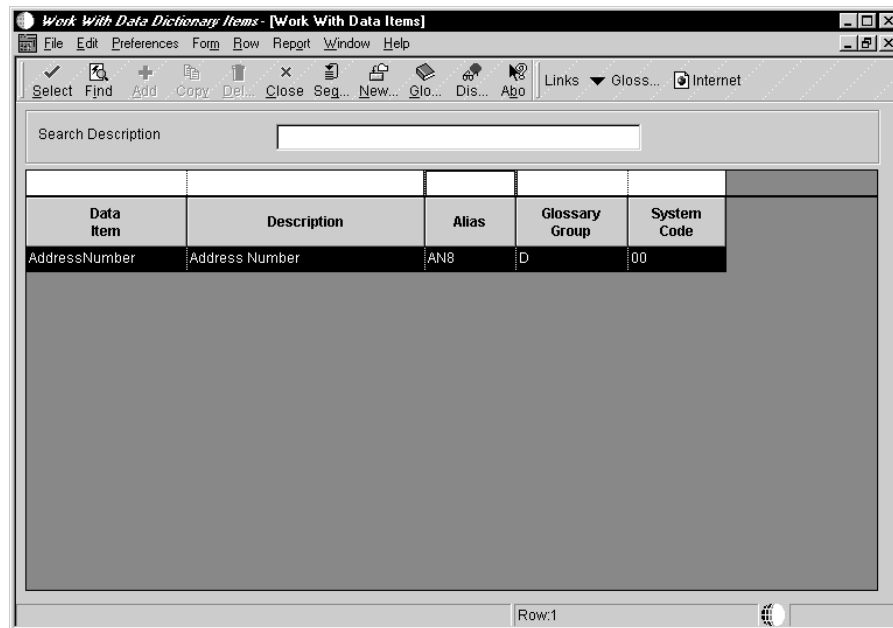
- Click Add and add a record for each language you support.

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

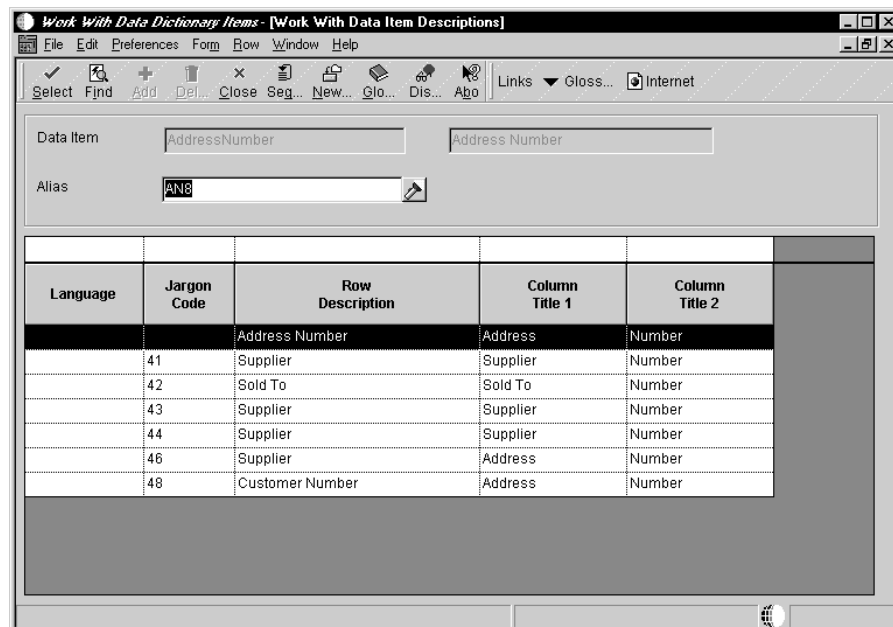
► To change row and column text for all applications

On Work with Data Items

1. Find the item you wish to update.



2. From the Row menu, choose Descript. Overrides



3. Choose a language record and modify it.

Language	Language Description	Jargon Code	Jargon Description	Form Name
Domestic Language	Domestic Language			W0401A
Domestic Language	Domestic Language			W3121F
Domestic Language	Domestic Language	12	Fixed Assets	
Domestic Language	Domestic Language	15	Property Management	
Domestic Language	Domestic Language	19	Utility CIS	
Domestic Language	Domestic Language	41	Inventory Management	
Domestic Language	Domestic Language	43	Procurement	
Domestic Language	Domestic Language	97	Software Installation	

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

For information about changing the row and column text for a particular application (interactive or batch) that use this data item refer to the *Package Management Guide*.

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 with the applications affected. This deletes the existing row and columns that are stored in a cache on the workstation.

Attaching Default Triggers

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

There are several specific types of triggers you can attach, including:

- Attaching a default value trigger
- Attaching a visual assist trigger
- Attaching an edit rule trigger
- Attaching a display rule trigger
- Attaching a next number trigger
- Attaching a smart field trigger

► **To attach default triggers**

On Data Item Specifications

1. Click the appropriate tab to attach any of the following applicable triggers to a data item:
 - Default Value
 - Visual Assist
 - Edit Rule
 - Display Rule
 - Next Numbers
 - Smart Field

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

Field	Explanation
Default Value	Use this field to assign a default value to the data item if the field is blank. • The value entered for numeric characters must be the exact same length as the data item and preceded with zeroes. • If this is a user defined code trigger, verify that the value entered is valid for the attached user defined code.
Visual Assist	There are three types of visual assist: • Calculator - assigns a calculator to the field. Use this on all numeric data items where a user might need to type in an amount. • Calendar - assigns a calendar to the field. Use this on data items where a user might need to type in a date. • Search form - assigns a search form to the field. Use this on all fields where the valid values for the field are found in a file.
Edit Rule	Use this to assign edit rules and procedures to a data item. Edit rules specify the editing technique that is applied when data is entered in the field. For example, value must be within a specified range. Use business functions to attach special logic that cannot be done through one of the edit rules. For example, if you would like to automatically verify the Address Book file every time you enter a Customer Number, then attach a procedure that says, Edit Address Book.

Field	Explanation
Display Rule	Use this to assign rules and procedures to a data item that governs how it is displayed. Display rules are keywords that describe a formatting technique that is applied when data is displayed. Use business functions when you need special processing that cannot be done through one of the five display rules. For example, if you would like to format the appearance of the account number, attach a business function called Display Account Number.
Next Number	Use this to assign next number logic to this data item. These rules include which system code is used to locate the next number and the array index number the data will use in the Next Number file (F0002).
Smart Field	This trigger is available for items in Glossary Group K and defines the business functions to be used to calculate a value and for column headings displayed on a report (using the reporting tool).

Attaching a Default Value Trigger

A default value trigger automatically inserts a specified default value in a field when:

- Control is Exited
- Dialog is Initialized

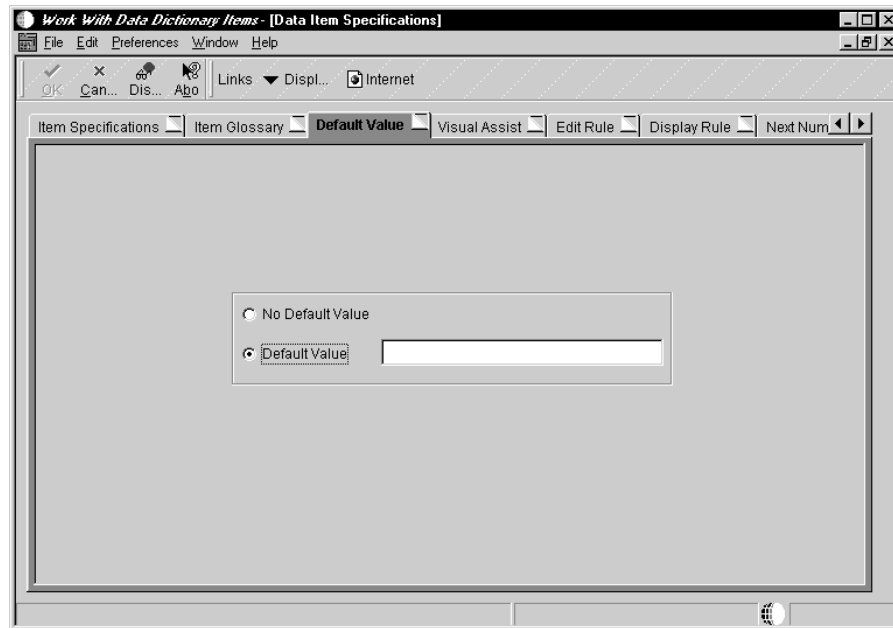
You use the default value trigger for fields where the default value is appropriate in almost every situation. If a value is passed to this field through a processing option, business function, form interconnection, data structure, or if the user types in a value, this default value will not be used.



To attach a default value trigger

On Data Item Specifications

1. Click the Default Value tab.
2. Click the following option:
 - Default Value
3. Enter the default value.



Field	Explanation
None Default Value	Indicates whether there is a Default Value on the field. Choose either of the following options: • None - no Default Value is assigned • Default Value - assigns a default value to the field. Specify the default value for entry that this field will have.

Attaching a Visual Assist Trigger

The following three types of triggers are available:

- Search form (flashlight)
- Calculator
- Calendar

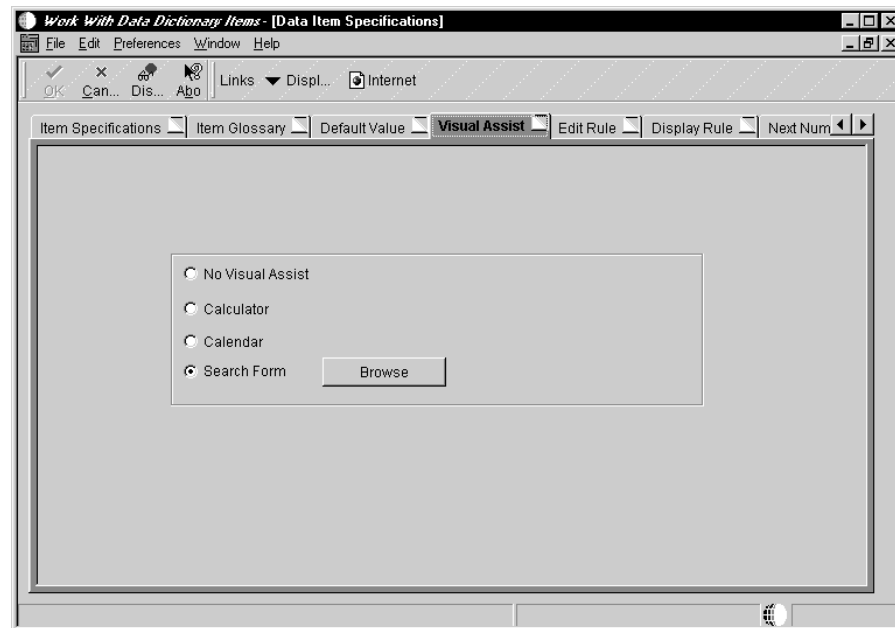
Use a search form trigger to link a field to a search form for locating valid values. The search form must exist before you attach a search form trigger.

► To attach a visual assist trigger

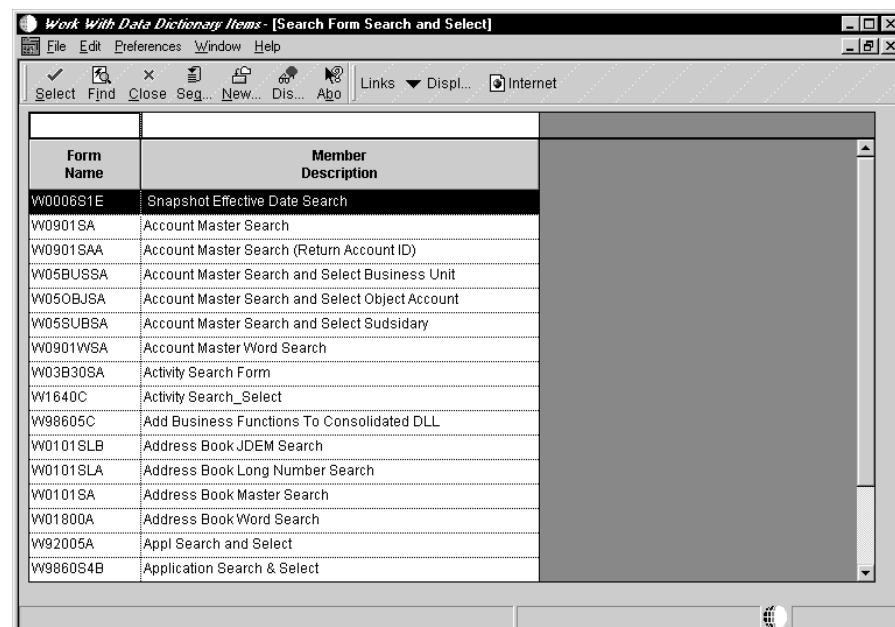
On Data Item Specifications

1. Click the Visual Assist tab.
2. On Visual Assist, click one of the following options:

- No Visual Assist
- Calculator
- Calendar
- Search Form



3. If you click Search Form, click the Browse button to select the form you want to access for valid values.



If you attached a Search Form assist trigger, at runtime when you click the Search Form flashlight button, the selected form appears. You can enter search criteria and return with a valid value. These Search and Select forms are based on files, not UDC tables.

Field	Explanation
Visual Assist	Indicates whether there is a visual assist on a field and the type. Choose from the following options: • None - no Visual Assist is assigned • Calculator - assigns a calculator to the field. Use this on all numeric data items where a user might need to type in an amount. • Calendar - assigns a calendar to the field. Use this on data items where a user might need to type in a date. • Search form - assigns a search form to the field. Use this on all fields where the valid values for the field are found in a file.

Attaching an Edit Rule Trigger

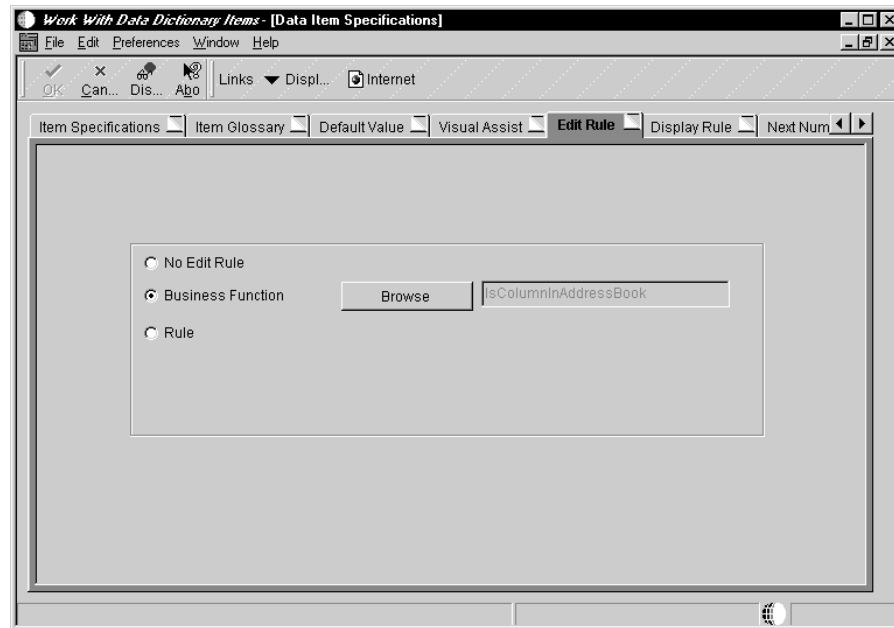
You use edit rule triggers to validate field values based on business functions or rules. For example, you can define a rule that:

- 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

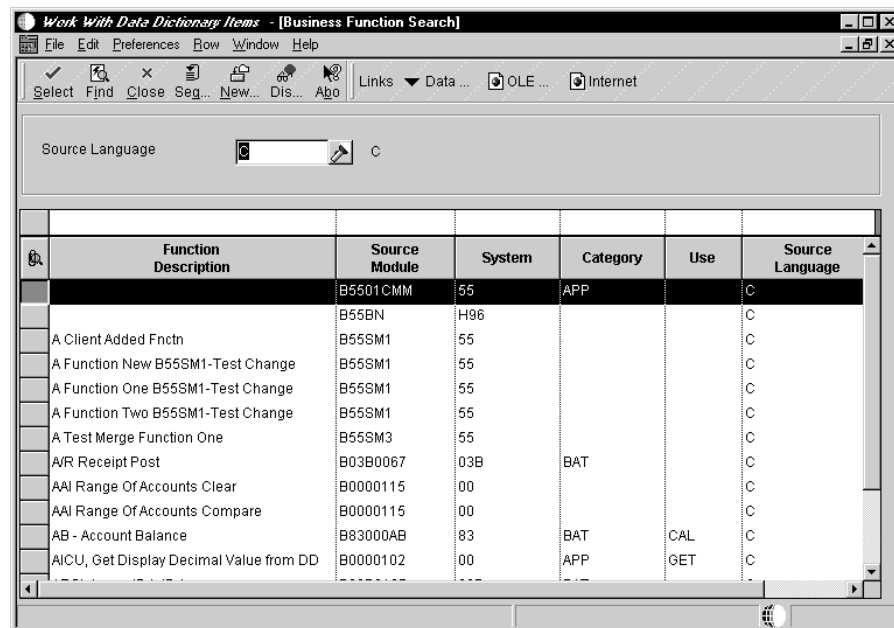
On Data Item Specifications

1. Click the Edit Rule tab.
2. On Edit Rule, click one of the following options:
 - Business Function
 - Rule



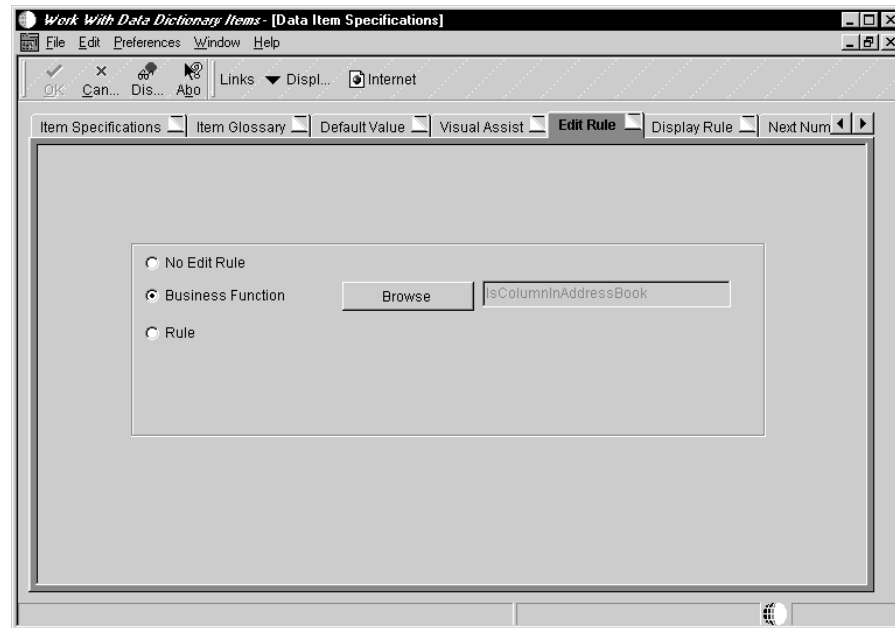
3. If you clicked the Business Function option, click the Browse button to choose from available business functions.

Create the business function if it does not exist.



4. Choose the desired business function.

Read the Attachments to learn what each function does.



5. If you clicked the Rule option, click the flashlight button to choose from the available rules.

These rules may 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
ZLNTH	Allocated Length (VARLEN flds)

Attaching a Display Rule Trigger

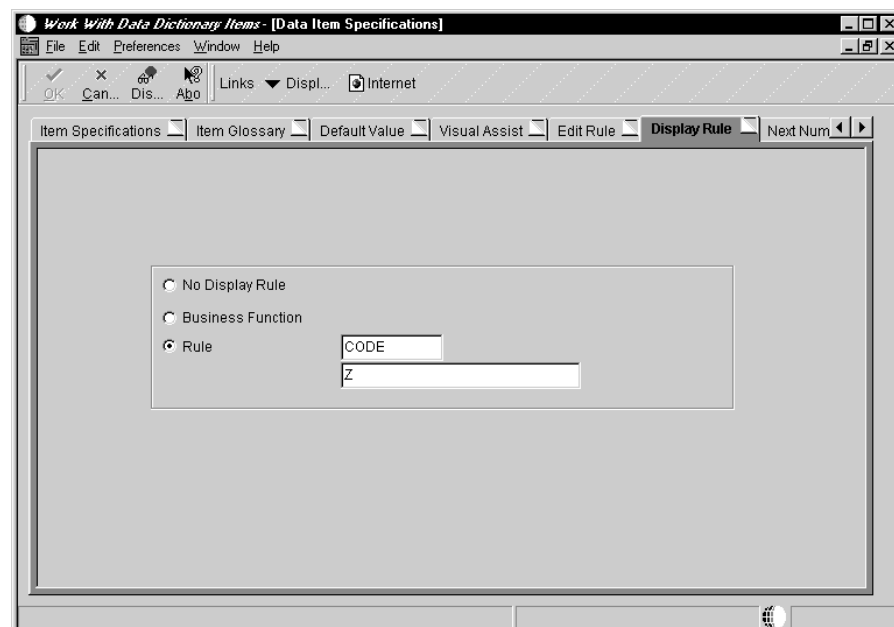
You use a display rule trigger to format data. You attach a display rule trigger based on either a business function or a user defined code. The following two types of display rule triggers are available:

- Business Function
- Rule

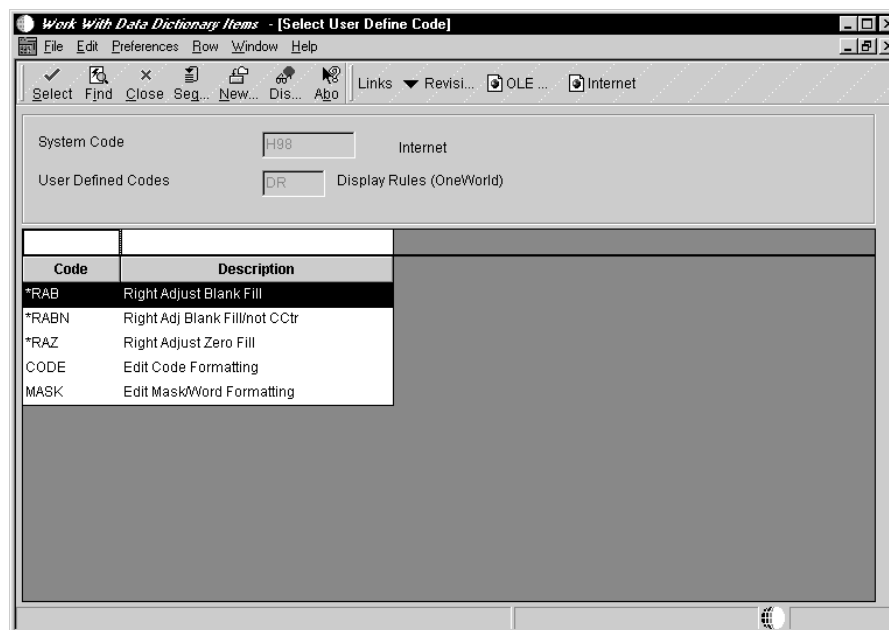
► To attach a display rule trigger

On Data Item Specifications

1. Click the Display Rule tab.
2. On Display Rule, click the following option:
 - Rule



3. Click the flashlight button for data display rules to browse and select from available user defined codes.



4. 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 the value and precedes it with zeroes. For example, Company would display as 00001. This is used for non-numeric fields only.

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 is displayed. For example, to display Social Security Number (SSN_) with embedded dashes, the mask parameter would show as:

bbb-bb-bbbb

Mask can only be used with char or string Data Item types.

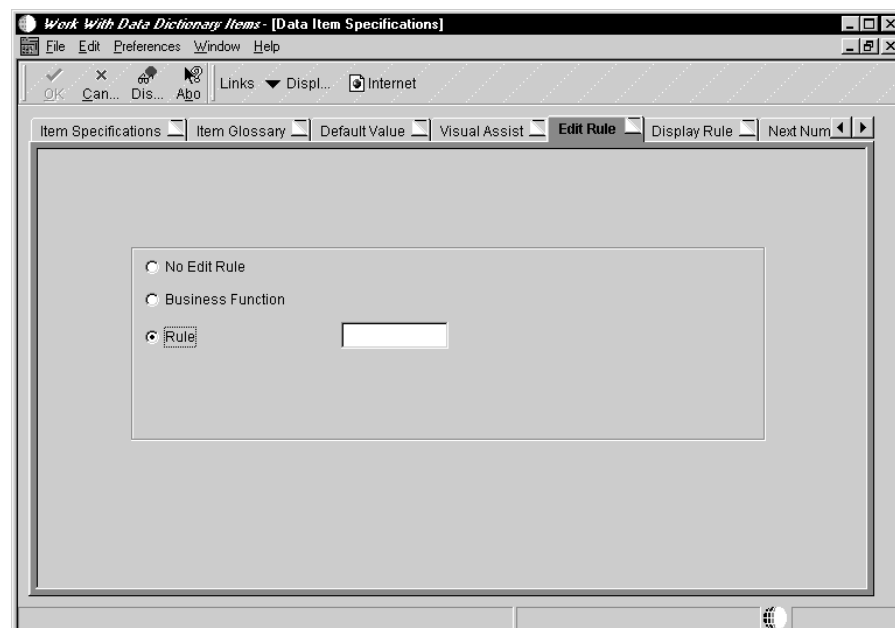
Field	Explanation
Formatting	Indicate whether or not there is a Display Rule for the field. Choose from the following options: • None - no Display Rule • Business Function - assigns a business function ID to the field. Specify the business function procedure in the field that becomes available when you choose this option. Use the visual assist to view and select a business function. • Rule - assigns display rule to the field. Specify the display rule in the fields that become available when you choose this action. Use the visual assist to view and select rules.

Attaching a User Defined Codes Trigger

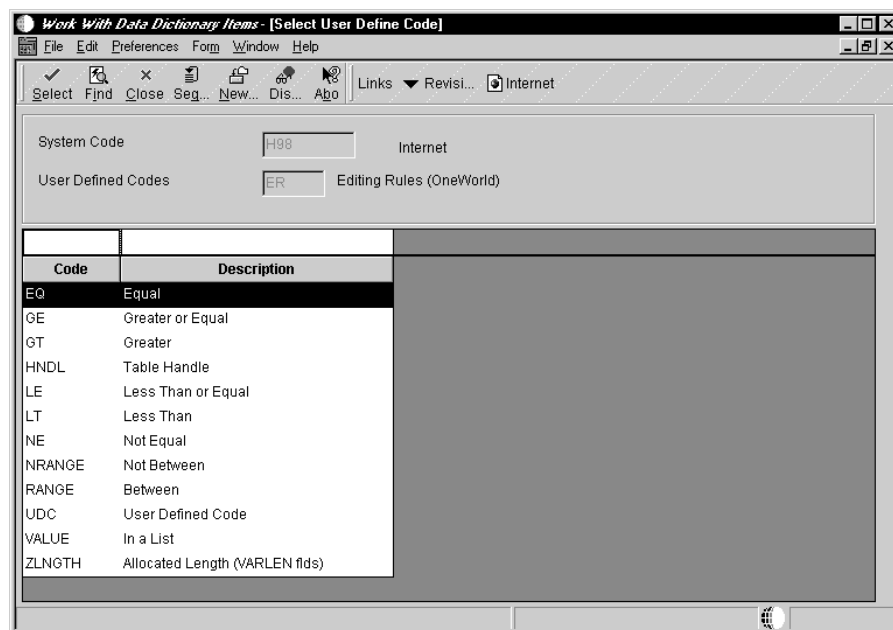
► To attach a User Defined Codes trigger

On Data Item Specifications

1. Click the Edit Rule tab.
2. On Edit Rule, click the following option:
 - Rule



3. Click the Rule flashlight button to browse and select UDC, User Defined Codes.



UDC is one of several choices that are available. Each option is described below:

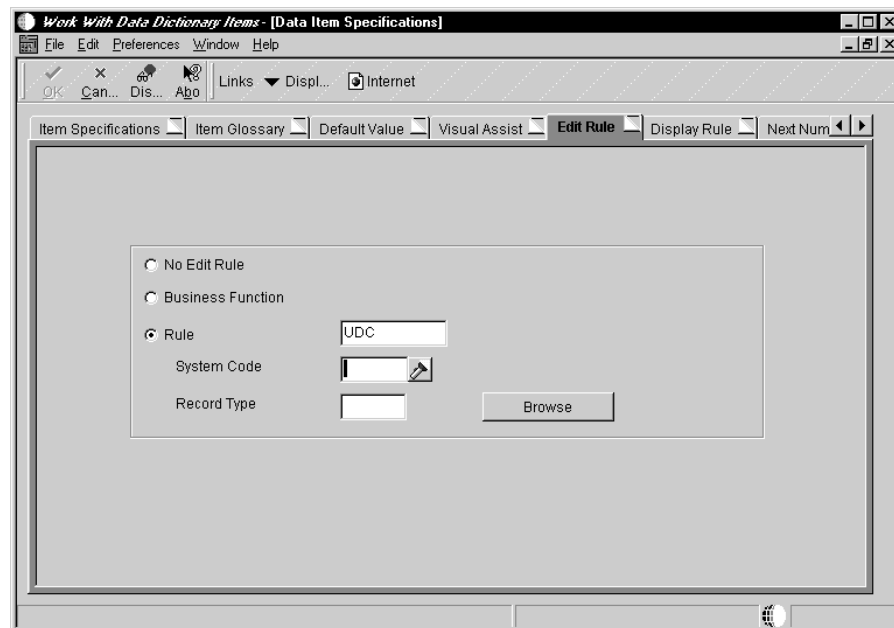
EQ	Equal to a particular value
GE	Greater than or equal to
GT	Greater than
LE	Less than or equal to
LT	Less than
NE	Not equal
NRANGE	Not in range
RANGE	Range
UDC	Link to a specific UDC Search and Select form
VALUE	When Y N 1 0 are the only valid values; Y N 1 0 will default into the parameter field.

ZLNTH

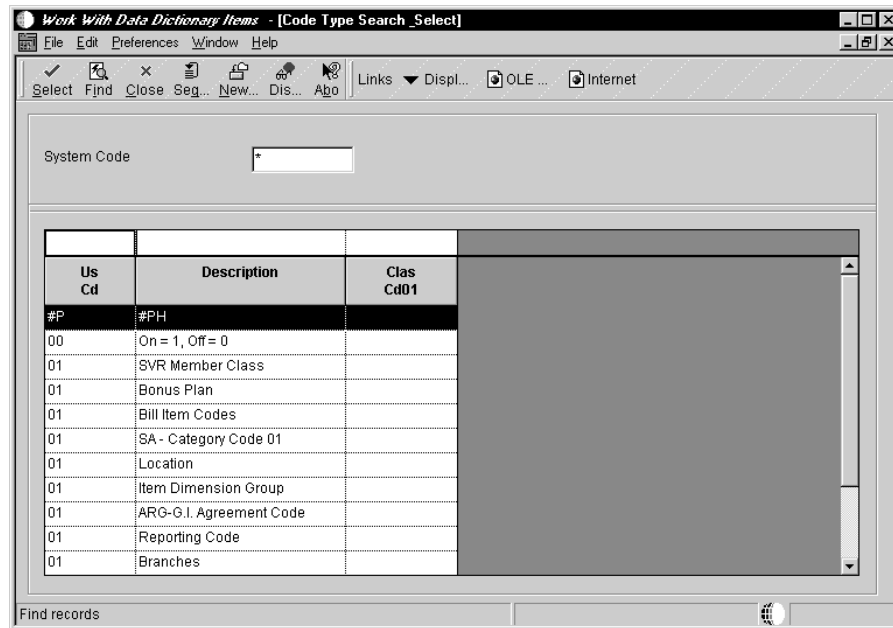
AS/400 database only. The parameter field contains the allocated length for a variable-length field. This specification is optional.

When you enter UDC in the Rule field, the following two fields are enabled:

- System Code
- Record Type



4. Click the system code flashlight to choose a system code.
5. Click the Browse button to choose a Record Type.



Field	Explanation
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.
Record Type	A code that identifies the table that contains user defined codes. The table is also referred to as a code type.

Attaching a Next Number Trigger

The next numbers facility controls the automatic numbering for such items as new general ledger account numbers, voucher numbers, and address numbers. It allows you to specify what numbering system code you want to use and gives you a method of automatically incrementing numbers to reduce transpositions and keying errors.

Use a next number when you want a numeric data item to default in a preassigned next number if the user does not type in 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 is F0002.

- 10 element array
- 1 record per system
- Modulus 11 check optional

After you set next numbers, do not change it. If you change next numbers, the following problems occur:

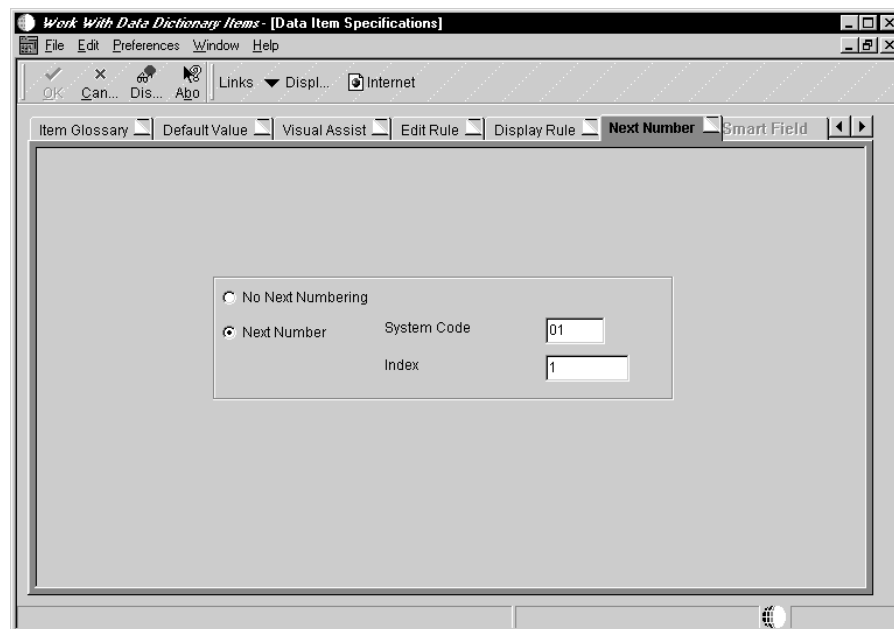
- System performance is impacted.
- Next numbers will not duplicate numbers; when it reaches the maximum, it starts over
- You cannot change position or add a new entry without programming modifications

Next numbers ties with the data dictionary. A data item in the data dictionary points to the next number system. You can view the next numbers facility by using the OneWorld Fastpath G00.

► To attach a next number trigger

On Data Item Specifications

1. Click the Next Number tab.
2. On Next Number, click the following option:
 - Next Number



3. Complete the following fields:
 - System Code
 - Index

Field	Explanation
Next Number	Indicate whether there is a Next Number on this field. Choose either of the following options: • None - no Next Number is assigned • Next Number - assigns next number logic to the data item. Specify the Next Number System Code and Next Number Index the data uses in the Next Numbers file.
System Code	Designates the system number for the Next Number retrieval. See User Defined Codes, system code '98', record type 'SY'.
Index	The array element number retrieved in the Next Number Revisions program. For example, the next voucher number is array element '02' of system '04'.

Attaching a Smart Field Trigger

Smart Fields are actually data dictionary items with attached business functions. The business functions that are attached include named mappings. This makes selecting a data item with particular functionality much easier. Instead of the end user having to know which business function to use and what parameters to pass, the user simply selects a data item that inherently has this information. Smart fields can be used for deriving a column heading or an object value in a tabular section in Report Design. Smart fields are always glossary group K.



To attach a smart field trigger

On Data Item Specifications

1. Click the Smart Field tab.

Work With Data Dictionary Items - [Data Item Specifications]

File Edit Preferences Window Help

OK Cancel Dis... Abo Links Displ... Internet

Item Glossary Default Value Visual Assist Edit Rule Display Rule Next Number **Smart Field**

Alias: ASSETAB
Account Balance

Smart Field Value BF: ABAccountBalance [Browse]

Smart Field Column Value Mapping: M12000AB

Smart Field Column Header BF: RollingColumnHeadings [Browse]

Smart Field Column Header Map: M12000CH

2. On Smart Field, complete one or more of the following fields:

- Smart Field Value BF
- Smart Field Column Value Mapping
- Smart Field Column Header BF
- Smart Field Column Header Map

On Smart Fields, you can click the Browse button to select a business function.

Work With Data Dictionary Items - [Business Function Search]

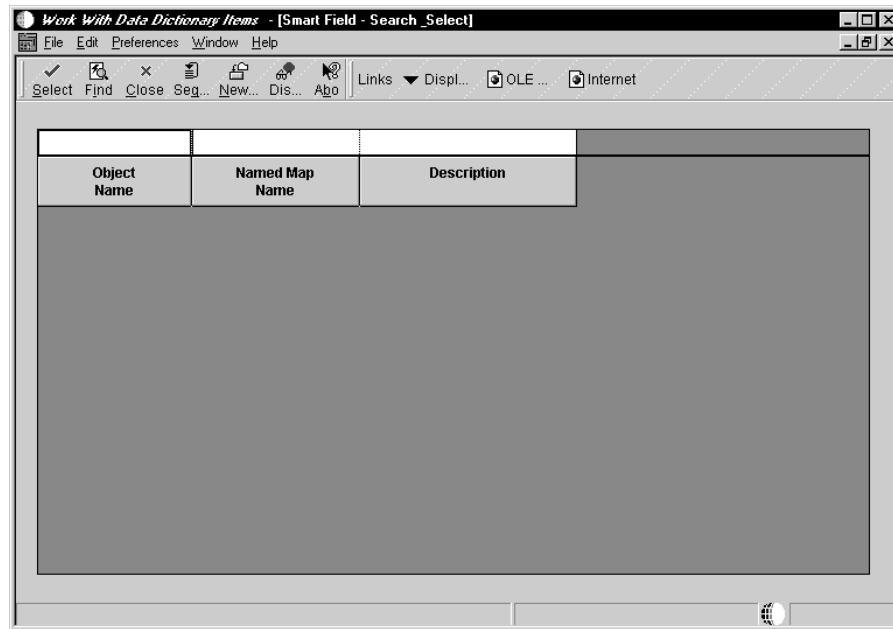
File Edit Preferences Row Window Help

Select Find Close Seg... New... Dis... Abo Links Data ... OLE ... Internet

Source Language: NER [Named Event Rules]

Function Description	Source Module	System	Category	Use	Source Language
	55TE	55			NER
A/P Draft Value Date Calculation - Japan	N7500050	75			NER
AAI Review to Update CHCD	N12920	12			NER
Access the Pay Cycle Parameters file - F072	N0700034	07			NER
Account Netting Begin Document	N03B0176	00			NER
Account Netting Clear Document	N03B0176	00			NER
Account Netting Edit Document	N03B0176	00			NER
Account Netting Edit Line	N03B0176	00			NER
Account Netting End Document	N03B0176	00			NER
Account Netting Summarize Document	N03B0176	00			NER
Account Structure Drag-Drop	N1000019	10	APP	UPD	NER
Accounts Receivable Billed/Paid Retrieval	N150012	15			NER

On Smart Fields, you can click the visual assist on Smart Field Column Value Mapping to select a value.



Updating the Glossary

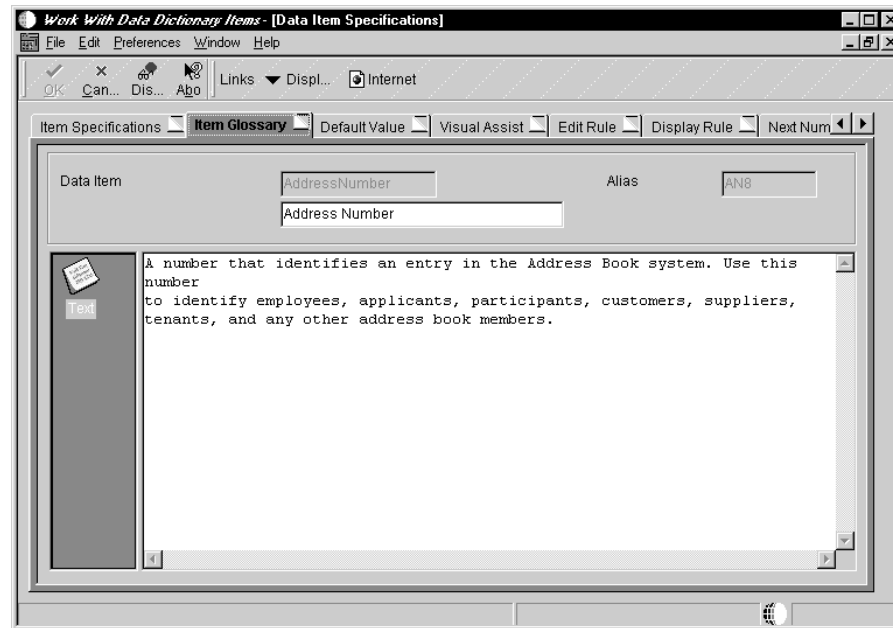
Each data item is described within the glossary. This text is seen by the end user of an application at runtime. Thus, it should explain how the field is used in the application. Within an application, you can access this glossary description in field-sensitive help. You can also:

- Add glossary text for new data items
- Create form-specific or system-code-specific glossary text
- Create glossary text for different languages
- Customize J.D. Edwards glossary text to suit your needs

► To update the glossary

On any form in Data Dictionary

1. From the Form menu, choose Glossary Data Item.
2. From the Row menu, choose Glossary.



3. Complete the following field:

- Glossary

Field	Explanation
Glossary	Updates the glossary description that appears when field-sensitive help (F1 or on-line help) is requested for this data item field within an application. This description may be customized to suit your environment.

Entering glossary text for a form displays that glossary description as additional information when you are on that form only.

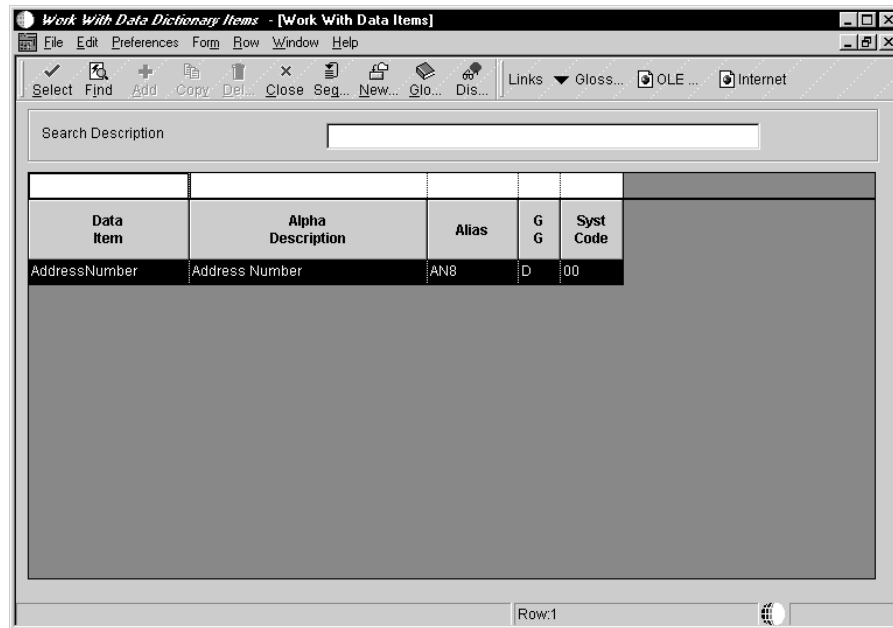
Adding Glossary Text for Languages

You can add separate glossary text for a new or existing item for different languages. For example, you can create glossary text for the base English language item and also have glossary text for French, Spanish, and German.

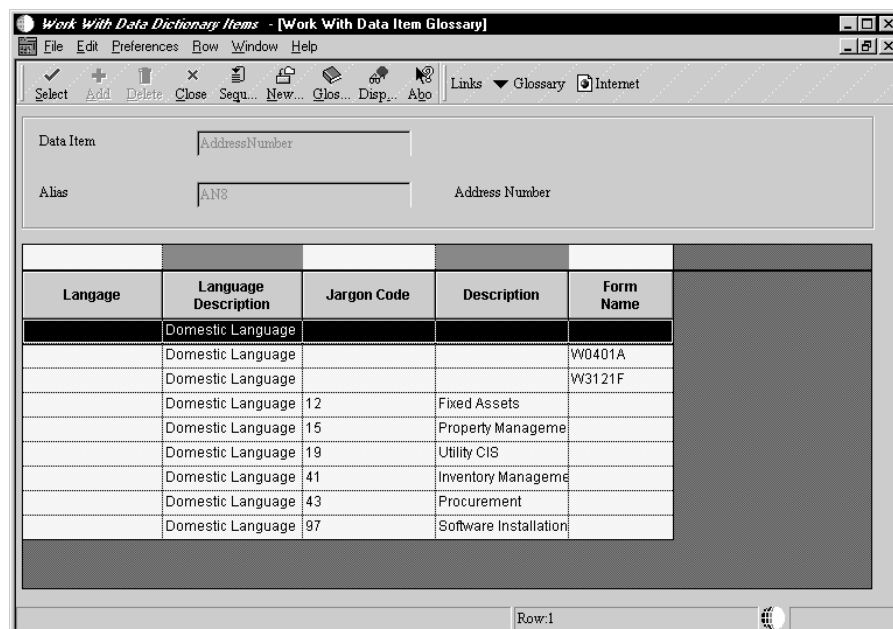
► To add glossary for an item with languages

On Work with Data Items

1. Choose the item you wish to change.



- From the Row menu, choose Glossary Overrides.



- On Work with Data Item Glossary, click Add.

Work With Data Dictionary Items - [Data Item Glossary]

File Edit Preferences Window Help

OK Cancel Dis... Ab... Links ▼ Displ... OLE ... Internet

Data Item: AddressNumber Alias: AN8

Language: Domestic Language: Form/Report: W3121F

Jargon Code:

This address book number corresponds to the address book number entered in the AN8 field on the work order header.

4. Type in the language for which you wish to create glossary text.
5. Type in a Form Name if you want the glossary to be for a specific form only.

Otherwise the glossary is for all forms that use this item.

6. On Work with Data Items, choose the row you just added.

Work With Data Dictionary Items - [Work With Data Item Glossary]

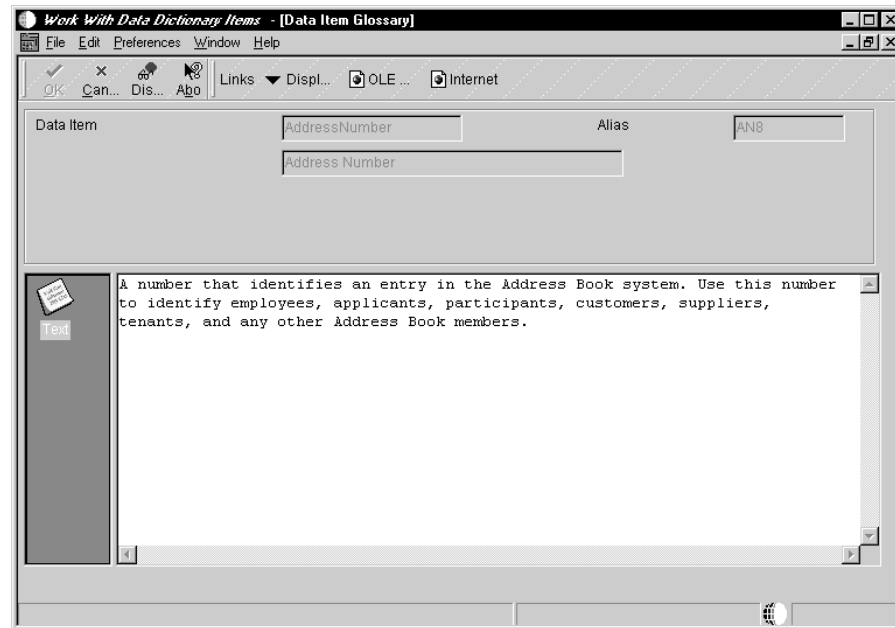
File Edit Preferences Row Window Help

Select Add Del... Close Seg... New... G Links ▼ Gloss... OLE ... Internet

Data Item: AddressNumber Alias: AN8 Address Number

L N	Lang Desc	Application Override	Description	Form Name
	Domestic Lar			
	Domestic Lar			W0401A
	Domestic Lar			W3121F
	Domestic Lar	12	Fixed Assets	
	Domestic Lar	15	Commercial Prope	
	Domestic Lar	19	Utility CIS	
	Domestic Lar	41	Inventory Managerr	
	Domestic Lar	43	Purchase Order Pr	
	Domestic Lar	97	Software Installatio	
F	French			

7. From the Row menu, choose Glossary.



8. Type in the glossary text you wish to add.

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



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 (the data items must exist in the data dictionary) for a table and assign key fields as indices for retrieving and updating data. You must define your table so that OneWorld recognizes that the table exists.

You must use Table Design to generate the table whenever you:

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

This section describes:

- ☐ Adding a Table
- ☐ Working with Table Design
- ☐ Viewing the Data in Tables

Tables

A table stores a set of data in columns and rows. Each column is a data item. Each row is a record.

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 faster.



Adding a Table

Determine whether an existing table contains the data items required by your application. If not, you must add a new table.

► To add a table

On Work with Object Librarian

1. In the Object Type field, enter TBLE.
2. Click Add.

3. On Add Object, complete the following required fields:
 - Object Name
 - Description
 - System Code
 - Column Prefix

See the J. D. Edwards Naming Standards below.

4. Complete the following optional fields:
 - Reporting System Code
 - Object Use
 - SAR Number
5. Click OK.
6. From the Form menu, choose Design.

Now you are ready to use Table Design to design your new table.

J.D. Edwards Naming Standards

J.D. Edwards recommends you use the following guidelines 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**

F = data table

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

00 - OneWorld 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

Provide up to a 60-character description for a table.

The table description is the topic of the table. If it came from the AS/400, it should be the same name as the file it represents, such as Address Book Master (F0101) and Item Master (F4101).

Indices

List the field as the index name, such as Address Number, if there is only one field in the index.

For coexistence, it is critical that OneWorld indices match logicals on the AS400. When you run the Generate Table command in Table Design, OneWorld automatically looks to the AS400 and checks to see if a matching AS400 file exists. If a matching AS400 file does not exist, then OneWorld creates logical files on the AS400. If a matching AS400 file exists, OneWorld does not create any logicals on the AS400.

If there are two fields in the index, list them consecutively, such as Address Number, Line Number ID.

List the first two followed by an alpha character (A), such as Address Number, Line Number, A if there are more than two fields in the index and the first two fields are the same as the first two fields of another index. Otherwise list the fields followed by a (+), such as Item Number, Branch, +.

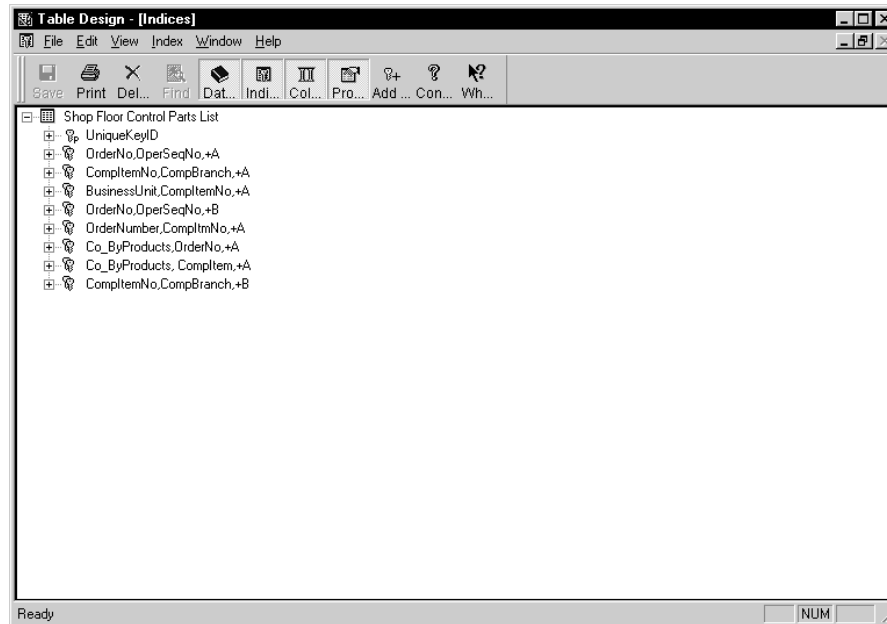
Place a comma (,) and space between each index field and between last index field and the plus sign.

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 will give you a warning of “Re-definition is not identical...”. This will impact fetches using the wrong index ID in business functions.

The following example shows multiple indices that have been defined for the Shop Floor Control Parts List table (F3111).

Example: Multiple Indices



Field	Explanation
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.
Object Use	Designates the use of the object. For example, the object may be used to create program, a master file, or a transaction journal. See UDC 98/FU.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.

Field	Explanation
Category Code – Object Librarian 06	The Data Structure Type indicates the type of structure as follows: 1 Function Data Structure 2 Processing Option Template It is edited on the 98/E6 UDC table.
Reporting System Code	A code that designates the system number for reporting and jargon purposes. Usually indicates the application module the object belongs to. See UDC 98/SY.
Column Prefix	A prefix associated with a particular table. The prefix is placed before the data dictionary data item name to give the field a unique name across J.D. Edward's systems. It usually comes from 0098FP. For non-coexistent tables, the column prefix is any two byte prefix. For coexistent tables, there must be a unique column prefix for each common table.
SAR Number	An abbreviation for Software Action Request (SAR). This number is used by the J.D. Edwards project control system to keep track of all charges and activity by individual SAR number.
Source Language	The source language code identifies the programming language that a business function is written in.
Process Type	The Process Type groups objects by operation, such as report, conversion, or batch process. It is edited on the 98/E4 UDC table.

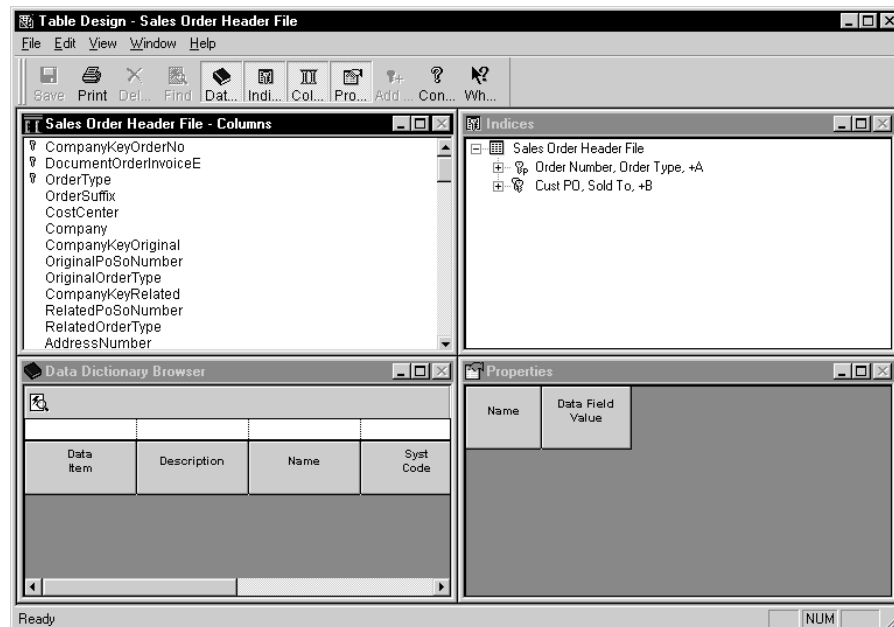
The following fields are typically required for an audit trail.

USER	User ID
PID	Program ID
UPMJ	Date Updated
JOBN	Workstation ID
UPMT	Time Last Updated

Working with Table Design

Table Design presents the following views within a single window:

- Table Columns displays the data items that make up your table.
- Data Dictionary Browser locates data items for selecting and moving to the Column view.
- Indices defines the unique data items for quicker sorting and updating of the table.
- Properties displays data item attributes for a selected data item in the Column view. This is a display view only and is primarily used when defining indices.



When you modify or delete data items or indices, you must reconfigure the table. Changes may 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 lost.



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 will produce error messages when you generate the application.

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

Use Table Design to complete the following tasks:

- ☐ Choosing data items for the table
- ☐ Defining indices
- ☐ Configuring tables

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

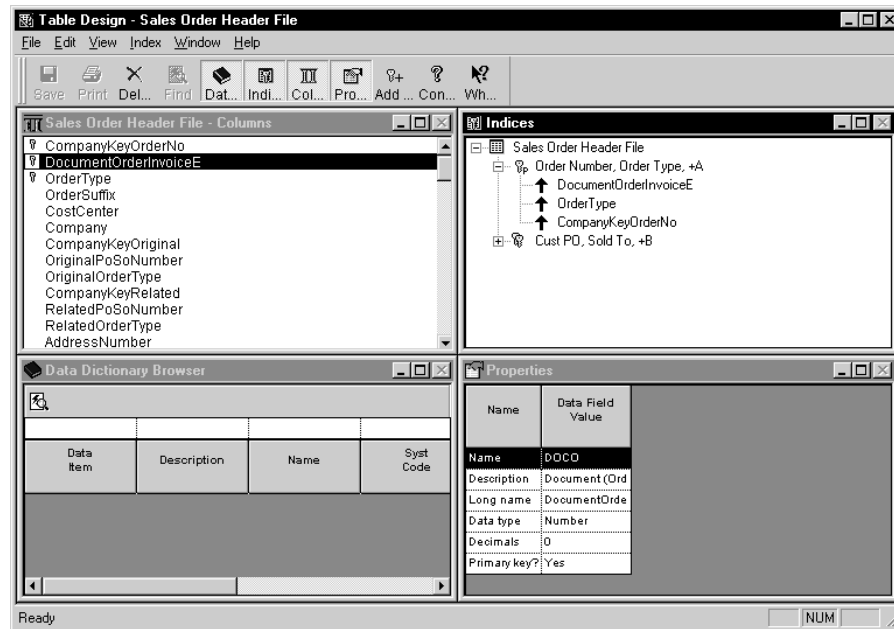
On Table Design Aid

1. Choose the table you wish to design.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1317270	SV4907030	8/6/96	00:00:00	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
U9311C	APPL_PGF	4	1317270	JAPAN03	7/30/98	10:40:08	

2. From the Form menu, choose Design.

- On Table Design Aid, in the Data Dictionary Browser view, use QBE to locate the data dictionary items you want to include in your table.



- To include a data item in your table, drag it from the Data Dictionary Browser view to the Table Columns view.
- To remove a column from a table, choose it and choose Delete from the Edit menu.

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 comprised 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.

OneWorld middleware generates an SQL statement that the native database understands.

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.

Refer to *Performance* in this guide for performance issues regarding indices.

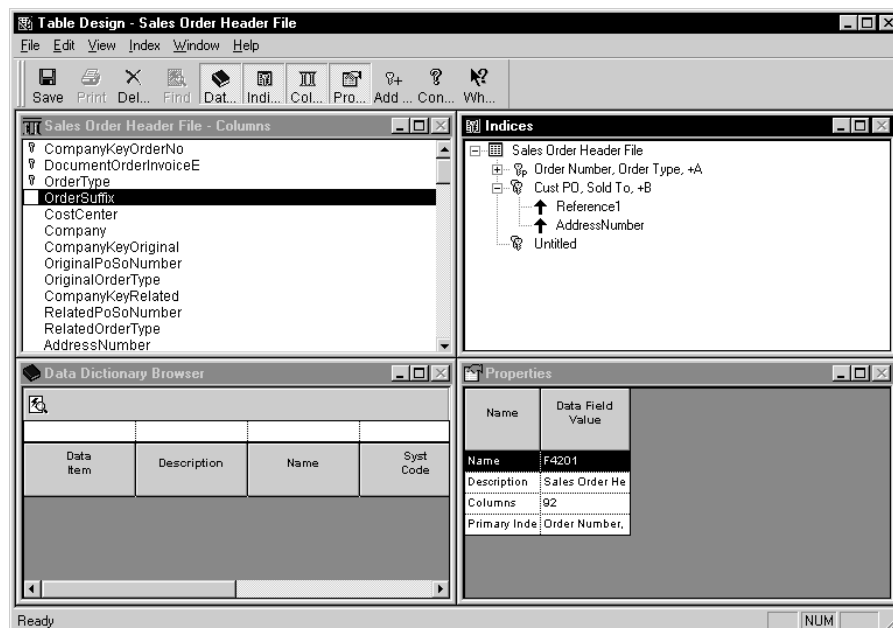
► To define an index

On Table Design Aid

1. Focus on the Indices form so the Index menu appears.
2. From the Index menu, choose Add.

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 to name the index. After you name the index, 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. Indicate the ascending/descending sort order for an index column. An arrow pointing in the upward direction indicates that the index column is sorted in ascending order.

J.D. Edwards Design Standards

Use the following guidelines to name an index:

- If there is only one field in the index, list just that field as the index name, such as Address Number.
- If there are two fields in the index, list them consecutively, such as Address Number, Line Number ID.
- If there are more than two fields in the index and the first two fields are the same as the first two fields of another index, list the first two followed by an alpha character (A), such as:

Address Number, Line Number, A

Otherwise list the fields followed by a (+), such as:

Item Number, Branch, +

- Place a comma (,) and space between each index field and between the last index field and the plus sign.
- You cannot have 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 will give you a warning of “Re-definition is not identical...”. This affects Fetches using the wrong index ID in business functions.

Refer to *Performance* in this guide for performance tips.

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 your generation will fail.

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

Table Design calls the Object Configuration Manager application (P986101) to configure tables. You can configure the table within any existing data source. If you do not indicate a data source, OneWorld automatically configures the table for the default map. You can change the path code to generate the table in a different location. This actually does a drop statement similar to the remove table, and then the table is recreated. If you regenerate the current table, the data in it will be lost.

You must regenerate a table if you modify it, for example if you delete or add data items. To ensure that data is not lost, you must export your data, generate the table, and copy the data back.

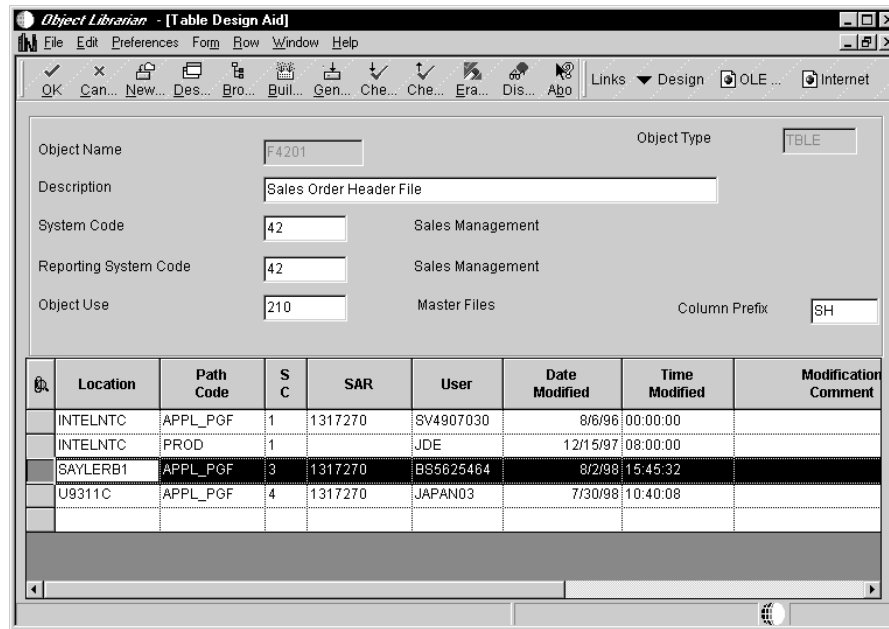
If you create additional indices, you must generate them. This modifies the .h file, but you will not lose existing data.

► To generate your table

On Table Design Aid

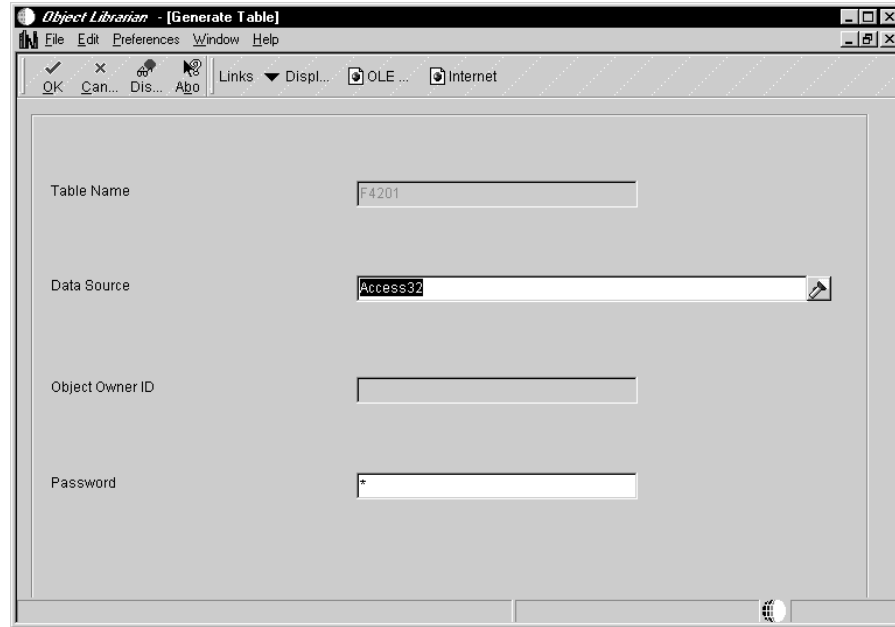
1. Save your table definition and exit Table Design.

OneWorld returns to the Table Design Aid form where you configure your newly defined table.

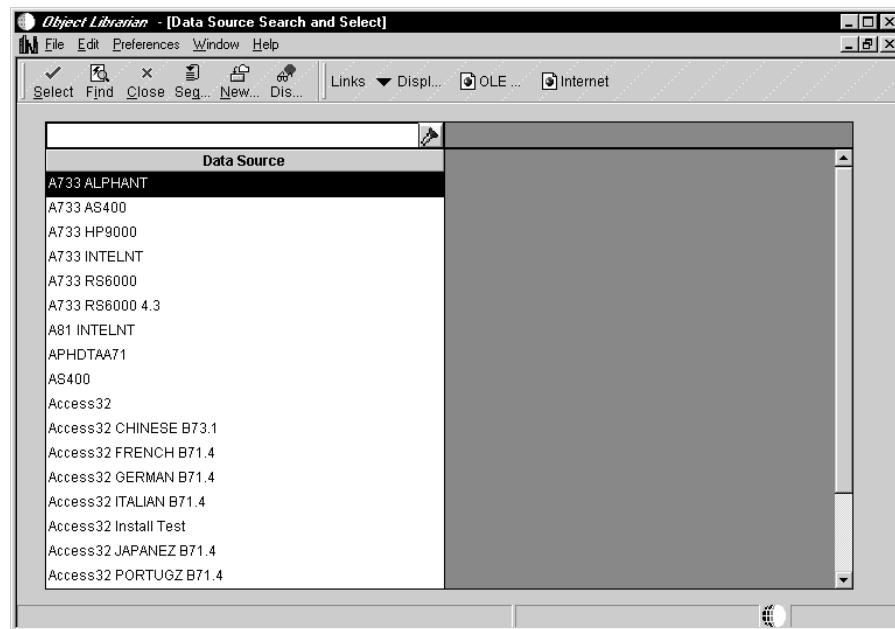


Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1317270	SV4907030	8/6/96	00:00:00	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3	1317270	BS5625464	8/2/98	15:45:32	
U9311C	APPL_PGF	4	1317270	JAPAN03	7/30/98	10:40:08	

2. From the Form menu, choose Generate Table.



3. Complete the Generate Table form and click OK.
4. Click the visual assist to access Data Source Search and Select.



5. Choose the data source you wish to use.

Object Librarian - [Table Design Aid]

File Edit Preferences Form Row Window Help

OK Cancel New... Des... Bro... Build... Gen... Che... Che... Era... Dis... Abo Links Design OLE... Internet

Object Name: F4201 Object Type: TABLE

Description: Sales Order Header File

System Code: 42 Sales Management

Reporting System Code: 42 Sales Management

Object Use: 210 Master Files

Column Prefix: SH

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1317270	SV4907030	8/6/96	00:00:00	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3	1317270	BS5625464	8/2/98	15:45:32	
U9311C	APPL_PGF	4	1317270	JAPAN03	7/30/98	10:40:08	

Table Creation Successful

A message appears at the bottom of the Table Design Aid form if your generation was successful.

Field	Explanation
Data Source Mode	Indicates whether OneWorld uses the primary or secondary data source. Valid values are: P Primary. S Secondary. You should only use this for business function (BSFN) data sources.
Data Source	The data source name.
Data Source	OneWorld uses this data source if the primary data source or the data item in the primary data source cannot be located.
Environment Name	For World, the Environment name is also called the Plan Name and is used to uniquely identify an upgrade environment for Install/Reinstall. For OneWorld (Install Applications), the environment name is also called the Plan Name and is used to uniquely identify an upgrade environment for Install/Reinstall. For OneWorld (Environment or Version Applications), this is the path code that identifies the location of the application or version specification data.
Override Application	Activates overrides for all objects used by an Application (currently unavailable).

Field	Explanation
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Allow QBE	Use this flag to turn ON or OFF row-level record locking for the data source. You should have this flag turned ON to help prevent database integrity issues. JDEBASE middleware uses this flag to determine whether or not to use row-level record locking.
Object Status	Indicates the data source called by the specified object.
User Class/Group	A profile used to classify users into groups for security purposes. Some rules for creating a User Class/Group are as follows: • The 'Class/Group' profile must begin with * so that it does not conflict with any System profiles. • The 'User Class/Group' field must be blank when entering a new group profile.

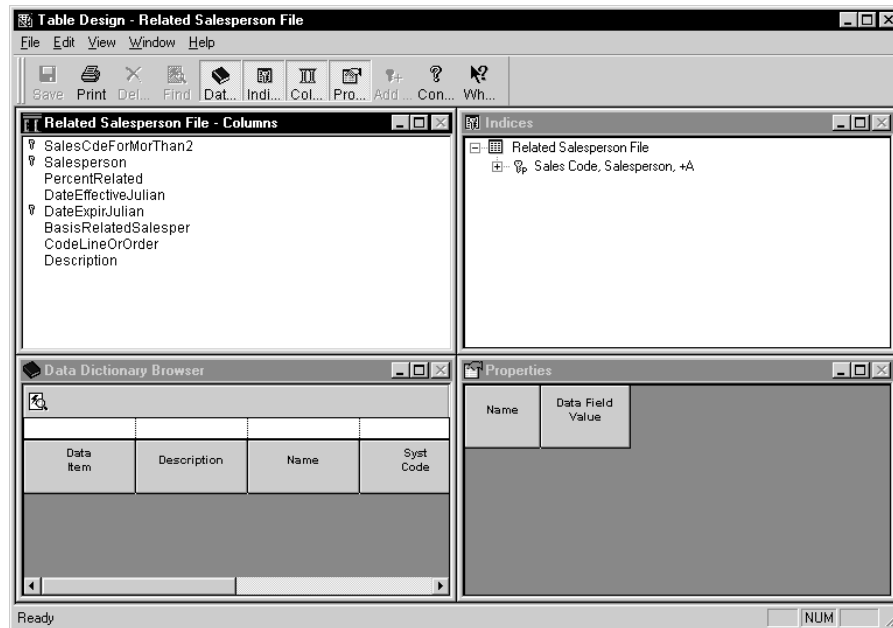
Previewing Tables

You can preview how tables will look when printed.



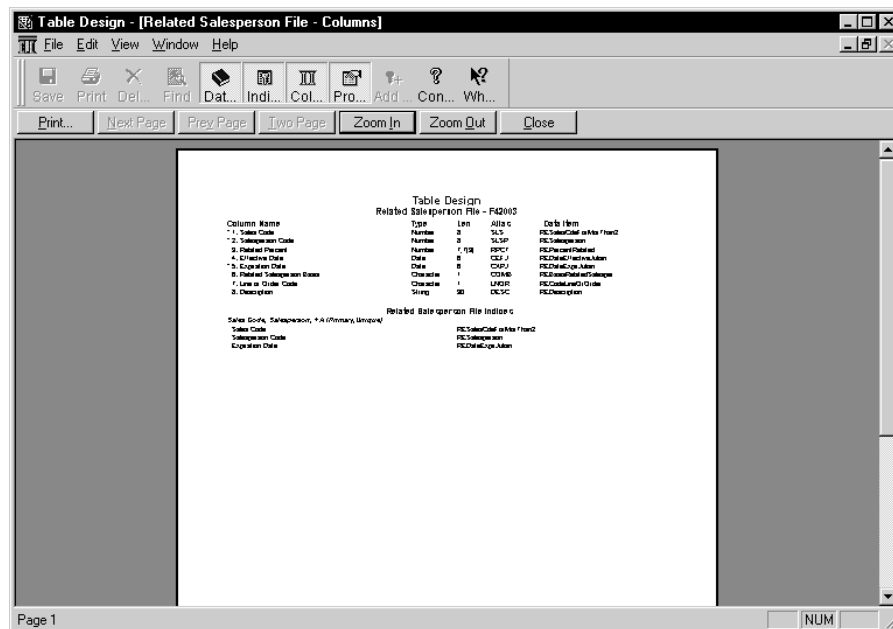
To preview your table

On Table Design Aid, from the File menu



1. Choose Print Preview

A print preview of your table displays.



Copying Tables

You can copy tables from one data source to another. This does not copy specs. You can also use table conversion to copy tables from one data source to another.

See Also

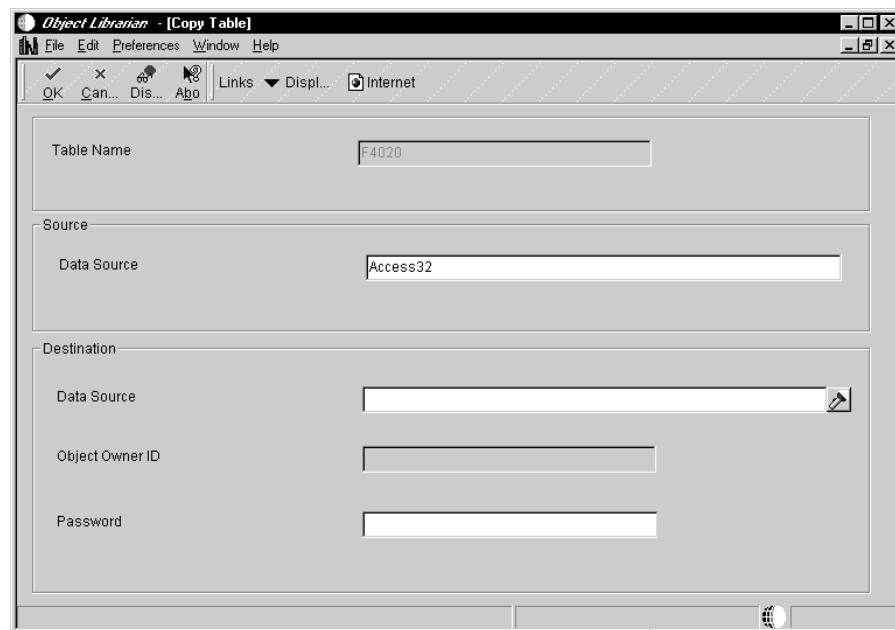
Table Conversion

► To copy your table

On Table Design Aid, from the Form menu

1. Choose Copy Table

Identify the data source you are copying from and to.



Viewing the Data in Tables

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

Complete the following tasks:

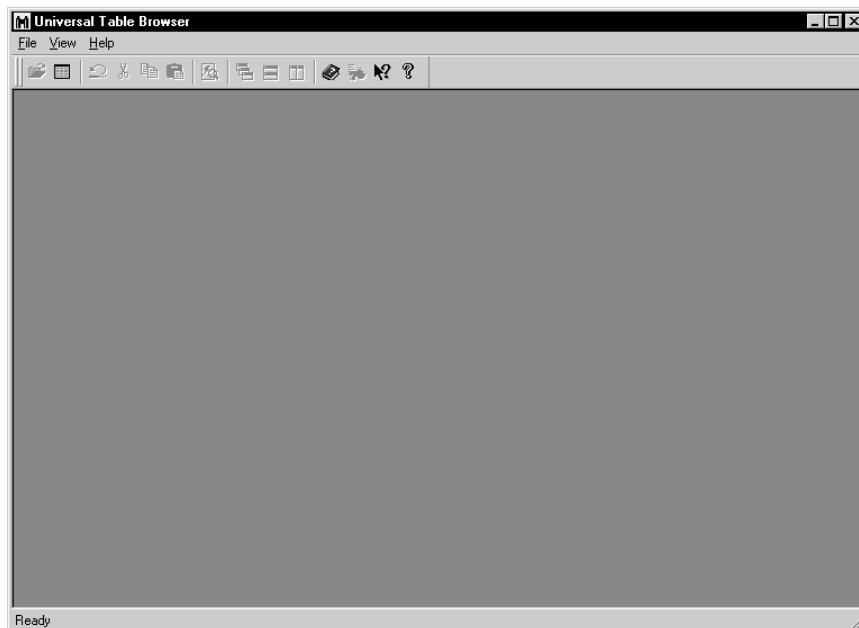
- View the data in tables
- View column properties in a table



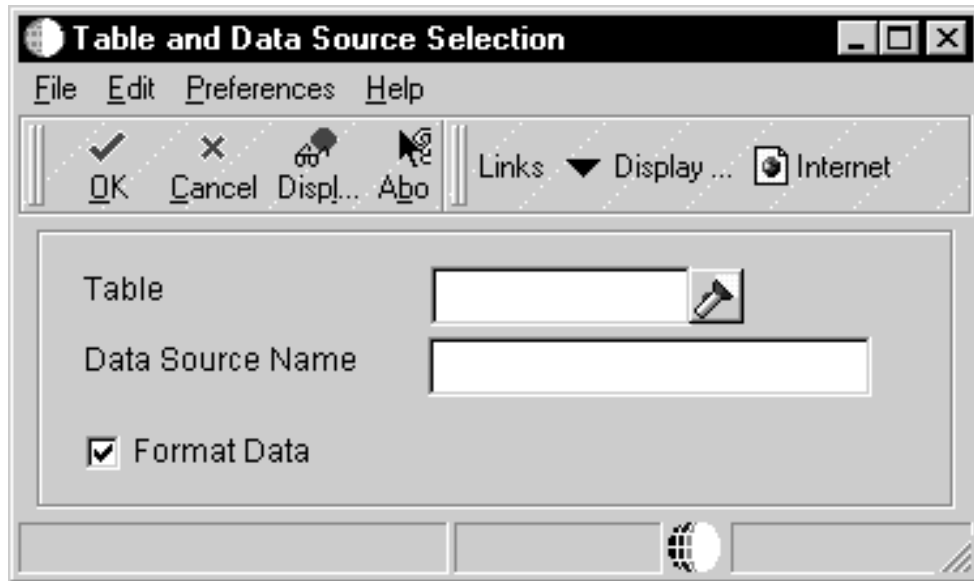
To view the data in tables

On Cross Application Development Tools (GH902)

1. Choose Universal Table Browser.



2. On Universal Table Brower, choose Open Table from the File menu.



3. Complete the following required fields:
 - Table
 - Data Source Name
4. Complete the following optional field:
 - Format Data

The Query By Example (QBE) function is operable as in any other OneWorld application. For example, you can enter ">50" in the ABAN8 column QBE to display records with an Address Book Number greater than 50. You could also enter "F*" in the ABALPH column QBE to display records with an Alpha Name that begins with the letter F.

The column sequence and column width functions are operable as in any other OneWorld application. You can rearrange the columns. For example, you might want to move a column that you use often from the end to the front, or move a column next to an associated column. You can also size the columns.

Field	Explanation
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures <i>Form-specific information</i> Indicates the name of the OneWorld table. For example, F0101 is the Address Book master. You can use the visual assist to enable the search and select form and locate a table.
Data Source Name	A valid data source in which the table resides. This default value is obtained from the OCM settings in the environment to which the user is signed on. Use the visual assist to enable the data search form and select any OneWorld data source.
Format Data	Indicates whether you want the Universal Table Browser to format portions of the data (default) or whether you want to view raw data. Formatted. The Universal Table Browser displays the data according to the specifications of the OneWorld data dictionary item. For example, assume that the data item PROC is a numeric field of size 15, with 4 display decimals. For a value of 56.2185, the Universal Table Browser displays a formatted value (using the data dictionary editing) as 56.2185, even though this value is stored in the database as 562185. Nonformatted. The Universal Table Browser displays the data according to the specification of the database and the data item type (such as numeric) from which the data came. For example, assume the table data item, PROC, is a numeric field stored in the database. Depending on the database, this field might default to a size of 32 with a precision of 15 being a numeric data type. Because OneWorld does not store the decimals in the database, a value 56.2185 would be stored and displayed in the database as 562185.0000000000000000.

Example: Universal Table Browser (formatted data)

In this example, a database table is shown as it was opened with the Format Data option turned on. 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	ABL
1			43.078.849/001	Financial/Distrib	FINANCIALDIST	1		
50				Project Managen	PROJECTMANAG	1		
60				Financial Report	FINANCIALREPC	1		
70				French Compan	FRENCHCOMPA	1		
77				Canadian Comp	CANADIANCOMF	1		
80				Colombian Com	COLOMBIANCOF	1		
200				Manufacturing/Di	MANUFACTURIN	1		
249				Model Energy & C	MODELENERGY	1		
1001			73.058.653/000	Edwards, J.D. &	EDWARDSJDCC	1		
2006			523785321	Walters, Annette	WALTERSANNNE	1		
2129			343298761	Jackson, John	JACKSONJOHN	1		
3001				Global Enterpris	GLOBALENTERI	1		
3002				Atlantic Corporat	ATLANTICCORP	1		
3003				CSC Corporatio	CSCCORPORAT	1		
3004				Pacific Company	PACIFICCOMPAI	1		
3005				Technology Syst	TECHNOLOGYS	1		
3333				Continental Inco	CONTINENTALIP	1		
3480				Digger In corpora	DIGGERINCORP	1		
4010				Colorado State T	COLORADOSTA	1		

► To view column properties in a table

On Universal Table Browser

1. View a table as described in *Viewing the data in tables*.
2. Right-click on a desired column and choose Column Properties.

If you are viewing a formatted table, the data dictionary properties are displayed in the upper-right portion of the Column Properties form.

If you are viewing an unformatted table, the data dictionary properties are not displayed.

Example: Column Properties

In this example, the column properties are shown for the OneWorld data dictionary item USEQ. The SQL database name for this OneWorld item is DTUSEQ.

The screenshot shows a 'Column Properties' dialog box with two main sections of input fields. The left section contains: SQLColumnName: DTUSEQ, Long Name: UserDefinedCodeSequence, Alias: USEQ, Size: 4, ID Dictionary: USEQ, System Code: 00, Data Type: EVDT_MATH_NUMERIC, Decimals Stored: 0, Decimals Displayed: 1, and Currency Column: 0. The right section contains: Glossary Group (empty), Can Have Security? (empty), Next Number System (empty), Search Form Name (empty), Edit Rule (empty), Display Rule: CODE:4, C Driver Type: JDEDB_C_DOUBLE, Offset in Buffer: 39, Actual Type: 8, Precision: 15, and Scale: 0. An 'OK' button is located at the bottom left of the dialog.

SQLColumnName:	DTUSEQ	Glossary Group	
Long Name:	UserDefinedCodeSequence	Can Have Security ?	
Alias:	USEQ	Next Number System:	
Size:	4	Search Form Name:	
ID Dictionary:	USEQ	Edit Rule:	
System Code:	00	Display Rule:	CODE:4
Data Type:	EVDT_MATH_NUMERIC	C Driver Type:	JDEDB_C_DOUBLE
Decimals Stored:	0	Offset in Buffer:	39
Decimals Displayed:	1	Actual Type:	8
Currency Column:	0	Precision:	15
		Scale:	0

OK

What You Should Know About

Using security in the Universal Table Browser

You cannot use OneWorld security to directly secure users from the Universal Table Browser, because it is a Windows executable application, not a OneWorld generated application. However, you can place form security on the Table and Data Source Selection form (W98TAMC). This effectively secures the Universal Table Browser, because the Windows executable cannot function without this OneWorld form.

All column and row security that you set up through Security Workbench apply to the Universal Table Browser. See *Working With Security Workbench*.

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 select only the data items that are required for your application. OneWorld uses the business view to generate the appropriate SQL statements needed to retrieve data from any of the supported OneWorld databases. After you have a business view, you can create a form that updates data (in an interactive application) or design a report that displays data. Because you select only those data items that are required in an application, there is less movement of data over the network.

A business view:

- Can contain data items from more than one table
- Can contain all or some of the data items in a table or tables
- Links a OneWorld application to a table or tables
- Is required to create an application or generate a report
- Defines the data items from multiple tables that an application uses (as in table joins or table unions)

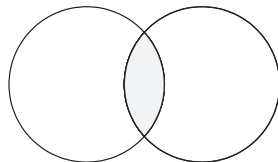
This section discusses the following:

- ☐ Adding a Business View
- ☐ Working with Business View Design

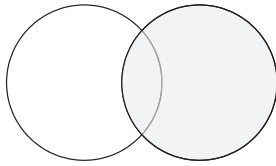
Table Join

A table join combines data from individual rows of the joined tables. The joining columns satisfy a join condition, such as when the rows have the same values in key columns. There are several types of joins, including the following:

- Simple Joins are also known as inner joins. They include only rows that match both tables.



- Right Outer Joins include rows common to both tables and unmatched rows from the right table.



- Left Outer Joins include rows common to both tables and unmatched rows from the left table.

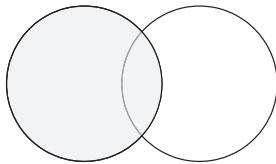
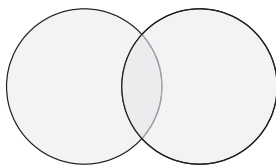


Table Union

A table union appends entire tables. It presents to the application selected rows from the first table first, then rows with corresponding columns from the second 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 first table and corresponding columns from the second table.



Select Distinct

You can use select distinct to help eliminate duplicate rows in the business view query.

Indices

The primary index from a table is used to build business views. Business views then carry information from the table to an application. If you need additional information other than the primary key carried forward to the application, you may need to change the business view index.

Adding a Business View

Before you begin designing a business view, think about the purpose of your application and the data items that it requires. Identify which OneWorld table or tables contain those data items. If you add a new business view, it does not impact performance. However, if you try to use an existing view that contains many more columns than you need, it may 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 should have the minimum number of items you need, so it is smaller. It should include only the basic fields needed for filtering and necessary output fields, such as descriptions.

Find/Browse and Parent/Child forms have a few more items and are more medium sized. They include those fields needed for filtering and necessary output fields, such as descriptions.

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



To add a business view

On Work with Object Librarian

1. Enter BSVW in the Object Type field.
2. Click Add.

3. On Add Object, complete the following required fields:

- Object Name
- Description
- System Code

See the J. D. Edwards Naming Standards below.

4. Complete the following optional fields:

- Reporting System Code
- Object Use

5. Click OK.

6. From the Form menu, choose Design.

Now you are ready to use Business View Design to design your new business view.

Refer to *Performance* in this guide for performance information about business views.

J. D. Edwards Naming Standards

J.D. Edwards recommends you use the following guidelines when you add a business view.

The Object Librarian name for a business view can be a maximum of 10 characters and should be formatted as follows: **VzzzzzzzA**

V = business view

zzzzzzzz = should be the characters of the *primary* table

A = letter to designate which view. For example V0101A is the first view over the table F0101, V0101B is the second view over the same table.

External Developer Considerations

External development refers to creation of applications by developers outside of J.D. Edwards, such as Partners in Development and J.D. Edwards consultants that might create custom applications for specific clients. You must use caution when naming a business view to prevent collisions between J.D. Edwards and non-J.D. Edwards objects. When creating a new business view over a standard J.D. Edwards table, format the business view name as follows: **Vssss9999**, where

V = business view

ssss = the system code applicable to the enterprise

9999 = a unique next number or character pattern unique 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.



There should be one business view for each table that includes 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 in the case of a header/detail form. In this instance, two business views may be selected, one for the header portion of the form and one for the detail portion of the form.

Joined Views

Format the name for joined views using 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

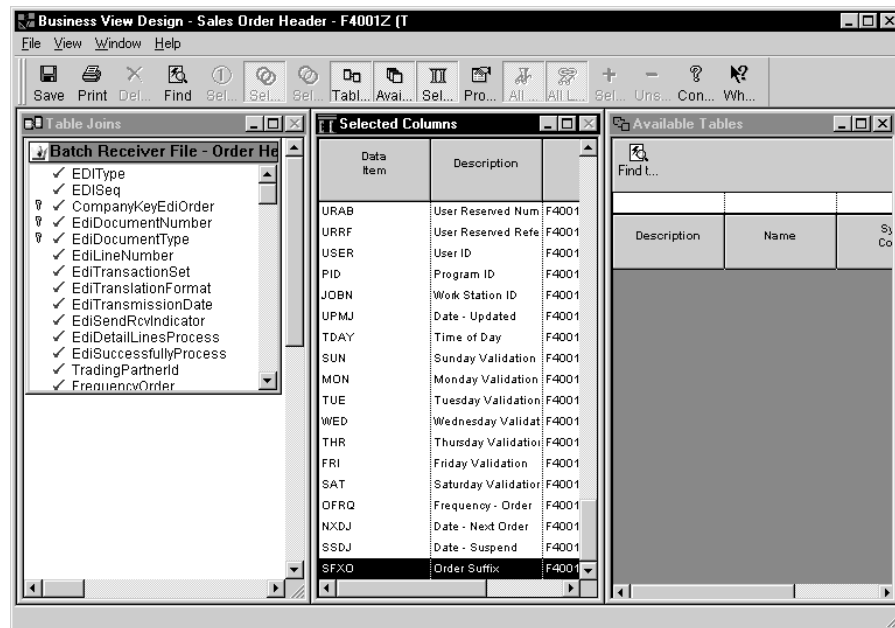
Field	Explanation
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.

Field	Explanation
Object Use	Designates the use of the object. For example, the object may be used to create program, a master file, or a transaction journal. See UDC 98/FU.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
SAR Number	An abbreviation for Software Action Request (SAR). This number is used by the J.D. Edwards project control system to keep track of all charges and activity by individual SAR number.
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.
Reporting System Code	A code that designates the system number for reporting and jargon purposes. See UDC 98/SY.

Working with Business View Design

The Business View Design tool presents several views within the same window:

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



You must generate the business view whenever you create a new business view or when you add a data item to the business view.

- When you delete a data item from a business view, if the database item is used in an application, an error results when you run the application.
- When you delete an entire table from a business view, any application that uses the business view will not run.
- If you delete a business view any forms that use the business view will fail.

This chapter describes the following tasks:

- ☐ Choosing a table for a business view

- ☐ Choosing data items for a business view
- ☐ Using Select Distinct
- ☐ Creating a table join
- ☐ Creating a table union
- ☐ Generating a business view

Choosing a Table for a Business View

Choose one or more tables from which to create the business view.

You can create a business view over one large table to retrieve and update only those columns needed by the application. However, performance is diminished when retrieving and updating a very large table. Consider joining two smaller tables rather than creating one large table that contains many data items.

To choose a table for a business view

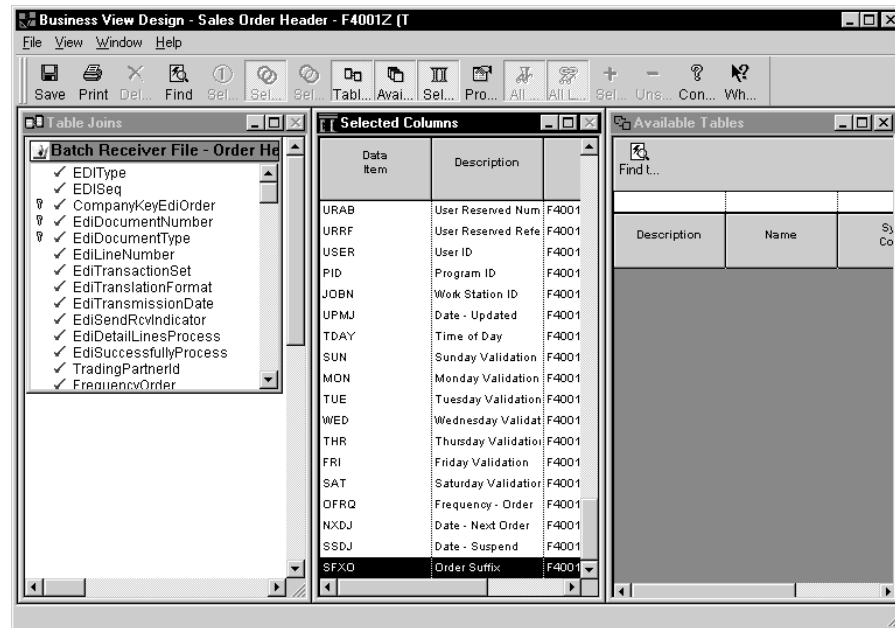
On the Available Tables view of Business View Design

1. Locate the table for the business view using the query-by-example line. You can search by Description, Name, and System Code.
2. Choose one or more tables and drag them to the Table Joins view. The view is called Table Join regardless of whether you are joining multiple tables or not.

The Table Joins view displays selected tables, along with the columns that comprise each. 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.



To ensure maximum performance in your application, OneWorld limits the number of tables to five if all joins are simple joins or to three tables if any of the joins is an outer join or if there is a union.



3. Designate a primary table by double-clicking the title bar of the desired table (optional).

If the business view contains multiple tables, the first table added is automatically designated 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 automatically the primary table.

4. To delete a table from a business view, choose it and choose Delete from the Table menu, or click the right mouse button and choose the Delete menu item.

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.



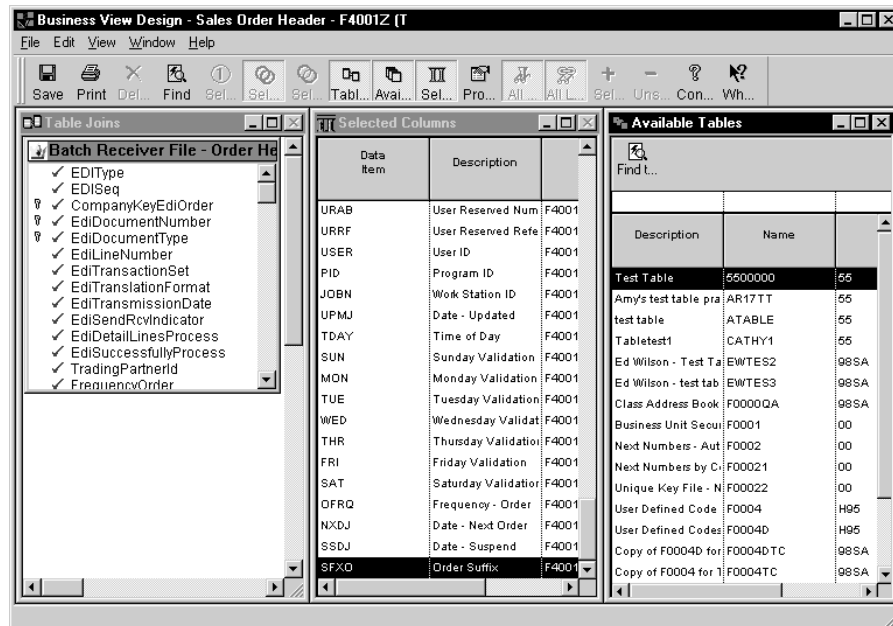
To choose data items for a business view

On the Table Joins view of Business View Design

1. Double-click the data items to include in the business view.

Selected data items appear highlighted in the Table Joins window. As you select each item, it is added to the Selected Columns window.

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



2. Double-click a selected item in the Table Joins window to remove it from the business view.

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 the same data item appears in multiple tables, you only need to select the data item from one table.

Do not select the same data item from two different tables; otherwise it will appear twice in 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 will be unique. The primary key will have 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 only need the first document number with Line 1, not all line numbers within the document. Select Distinct will fetch only the first occurrence of the document number, not all the line numbers within it.

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

CO	Company
CRCO	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

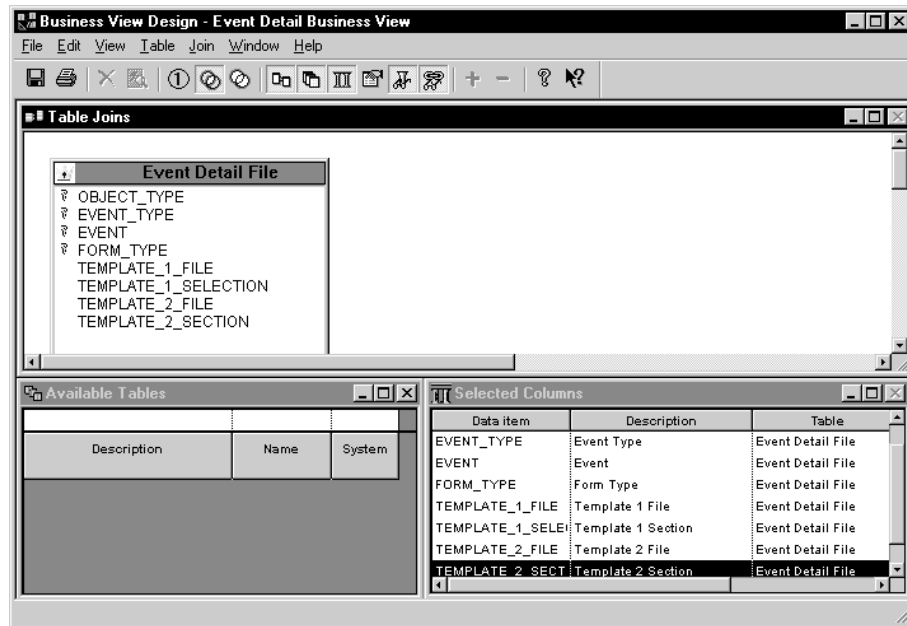
On Business View Design Aid

1. From the File menu, choose Distinct.
2. Choose the primary table of your view.
3. From the Table menu, choose Change Index to change the index of the primary table to a nonunique index.

Example: Select Distinct

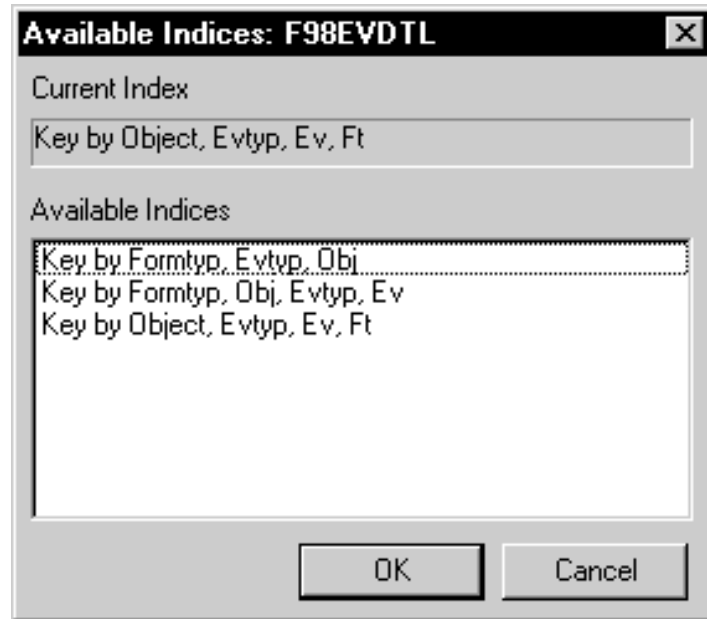
This example illustrates Select Distinct. The business view used for this example, V98EVDTL, uses the primary index of the primary table (in this example it is F98EVDTL) by default. The primary index of a OneWorld table must be unique. Because of this, duplicate values are not returned when the business view query is generated. Business View Design Aid also allows you to use any other index of the primary table when processing the business view.

Choose the primary table in Business View Design Aid. From the Table menu, choose Change Index to change the primary table index. Change Index is only available for the primary table.

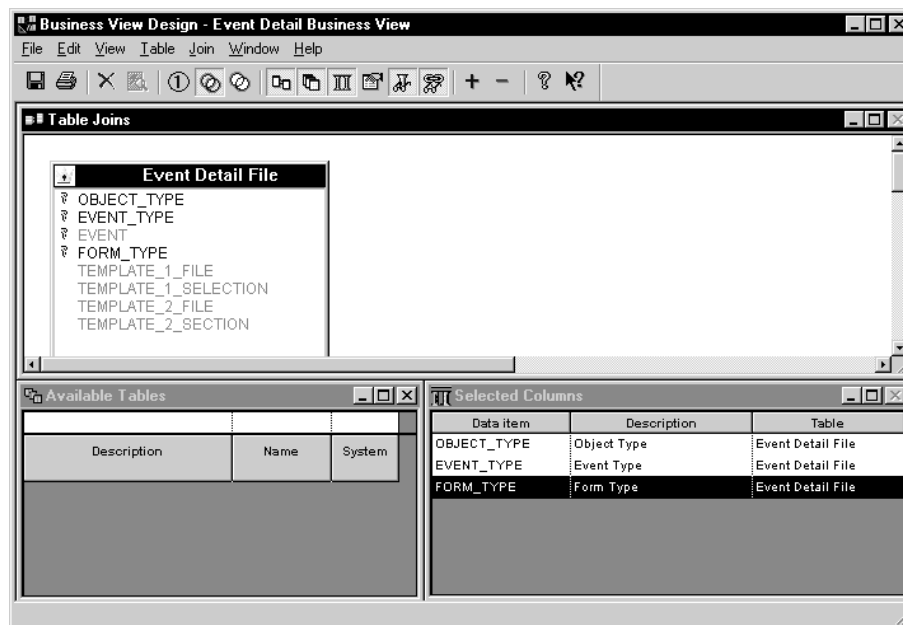


A warning indicating that your selected column list will be changed appears. Click Yes to continue.

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.



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



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

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.

Next you reopen the V98EVDTL business view. Choose Select Distinct from the File menu. Choose Change Index to reselect the Key by Formtyp, Evtyp, Obj index from Available Indices and click OK. Save the business view and exit Business View Design Aid.

At this point you may need to exit and restart OneWorld. OneWorld caches the business view in memory, so even though you have changed the business view, the previous cached business view will run until it is cleared out.

When you generate and rerun the same application using the V98EVDTL business view 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 53 rows of data from table F98EVDTL.

Creating a Table Join

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

Joins are typically used for non-input-capable forms, such as Find/Browse forms, and reports. They are not usually used 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, they must be linked by establishing joins between columns in those tables. The links instruct the system on how rows between a table correspond to rows in another table.

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

Look at 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 be sure your needs have been well thought out before you do so.

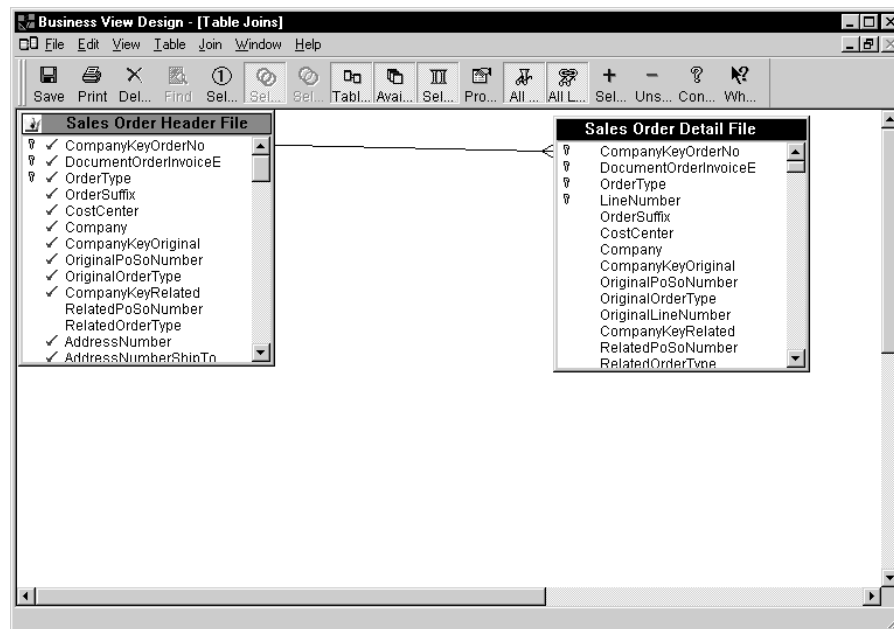
Ensure that the two columns you want to join are visible on the Table Joins window.

► To create a table join

On Table Joins

1. 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 like columns. The name 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, view the data item attributes that are displayed in the Hover Hints for each data item.



2. To delete a join, choose it and choose Delete from the Join menu, or click the right mouse button and choose Delete from the pop-up menu.
3. From the Join menu, choose Types and choose one of the following join types (the default is Simple):
 - Simple
 - Left Outer
 - Right Outer
4. From the Join menu, choose Operators and choose one of the following operators (the default is Equal)
 - Equal (=)
 - Less than (<)
 - Greater than (>)
 - Less or equal (<=)

- Greater or equal ($\geq$)
5. Click the Selected Columns view to sequence columns.

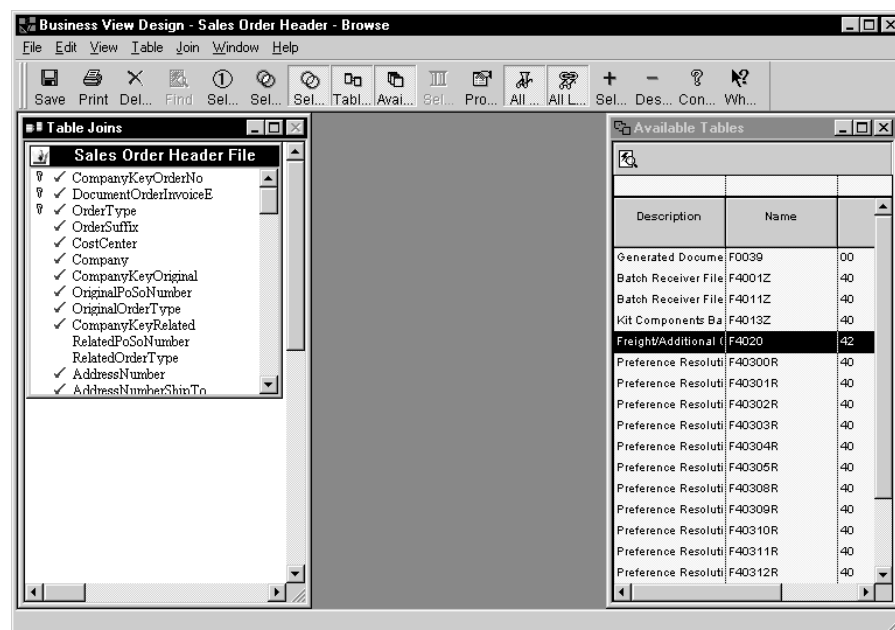
Creating a Table Union

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

On Business View Design

1. From the File menu, select Unions.
- The view will still say Table Joins.
2. Choose the tables you wish to unite.



Generating a Business View

After you have defined your business view specifications, you must save and exit the design tool and then generate the business view.

► To generate a business view

On the Business View Design tool

1. Save your business view definition and exit the design tool.

OneWorld returns to the Business View Design Aid form where you generate your newly defined business view.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1785727	KB5469772	7/14/98	13:11:09	
INTELNTC	PROD	1	1785727	MV5703469	7/14/98	15:07:56	
SAYLERB1	APPL_PGF	3	1785727	BS5625464	8/2/98	16:19:43	

2. On Business View Design Aid, choose Generate from the Form menu.

This creates a business view object based on the specifications you just defined. There is usually one business view that relates straight across to one application. Generate creates a header file for the business view. The header file is only good if you write a business function that uses the business view for input/output. If you use a business function you must perform the generation above.

If the generation was successful, a “Generation Successful” message appears. You can now use this business view in an interactive application or batch process.



Data Structures

Data structures are a key element of any programming language or environment. A data structure is a list of parameters that is used to pass data between applications and tables or forms. OneWorld uses data structures in the following instances:

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

System Generated Data Structures

Form Data Structures

Each form with an attached business view has a default data structure. Maintain the data structure using the Form/Data Structure menu option in Forms Design. This data structure is used to receive parameters from or send parameters to other forms during Form Interconnects.

Report Data Structures

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

User Generated 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 Librarian.

Processing Options Data Structures

Processing options are used to create an “input property sheet.” A parameter list is used to pass processing options to an application. You can create a processing option data structure template or modify an existing one in Object Librarian.



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. Accomplish this by creating a DSTR object type, or select an existing one to work with in Object Librarian.

You can also create data structures for text substitution messages (refer to the Messaging section in this guide).

You can attach notes, such as an explanation of use, to any data structure or data item within the structure.

This section describes the following topics:

- ☐ Working with Interconnect Data Structures
- ☐ Creating a Data Structure

Working with Interconnection Data Structures

Interconnection data structures are maintained by the Form Design or Report Design tool, not by Object Librarian. 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.

A default data structure is created at the report level for a report with an attached business view. The default data structure is empty. If data items are added to the data structure, the structure is maintained by Report Design, as is the report.

This chapter describes the following:

☐ Modifying form data structures

You create the default data structure by using keys from the business view selected for the form.

☐ Modifying report data structures

☐ Displaying type definitions

Modifying Form Data Structures

You can access the form data structure from Forms Design. From the Form menu, choose Data Structure. The Form Data Structures form is displayed.

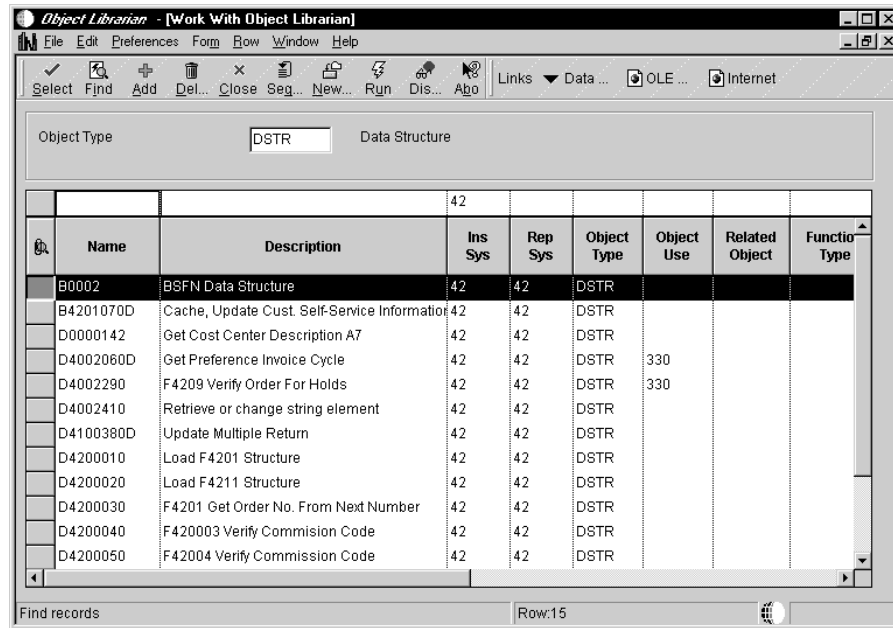
The items in the existing data structure are displayed in the Structure Members list. Two tabs appear for modifying the data structure. The Available Objects Tab lists data items in the attached business view. If you want to add an item to the data structure that is not in the Available Objects list,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

Depending on whether the item you want to add to the data structure is in the attached business view, use the following tabs:

- Available Objects tab
- Dictionary Items tab

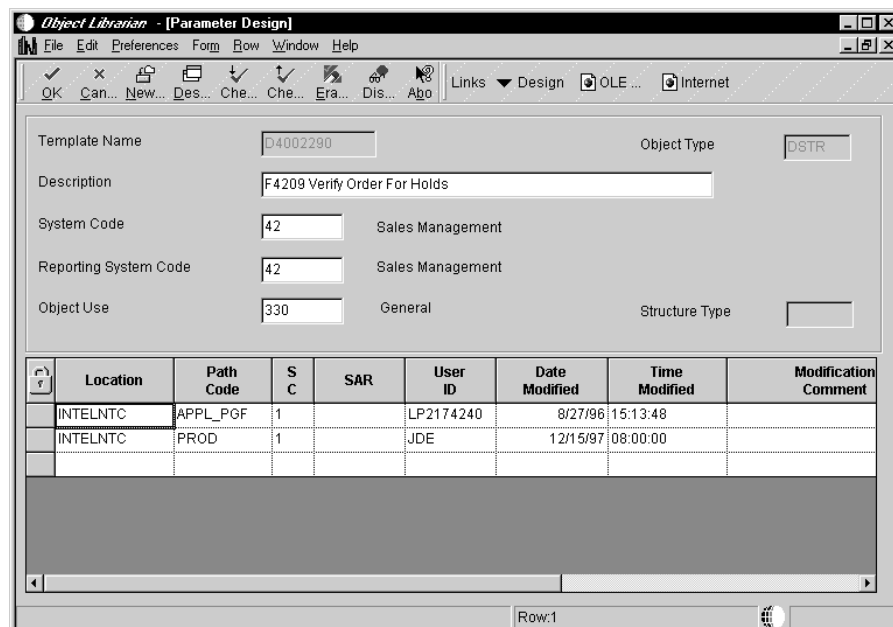
► To select from Available Objects

On Object Librarian



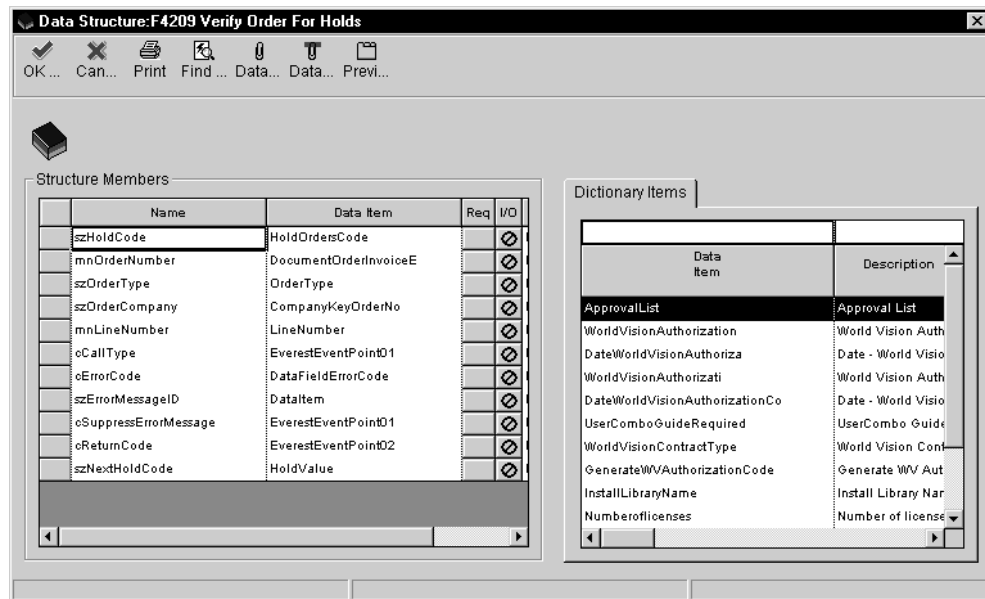
1. Choose the data structure you wish to modify.

The Parameter Design form appears.



2. On Parameter Design choose the data structure.

- From the Form menu, choose Design.



- On Data Structure, choose a data item from Dictionary Items list.

To more easily locate an item in the list, enter a Name, Alias, System Code or combination in the QBE fields and click Find.

- To change the name, double-click on the name and enter a new name. Use Hungarian notation rules.
- Drag the desired data items to Structure Members.
- You can indicate the required status by leaving the required field blank or choosing one of the following:
 - A green X to indicate the field is optional.
 - A red check mark to indicate the field is required.
- Choose a direction for the data flow.
- Drag the desired data items to Structure Members.
- Click OK.

The status line displays any error messages.

You can use the note attachments to add information about your interconnections.

Modifying Report Data Structures

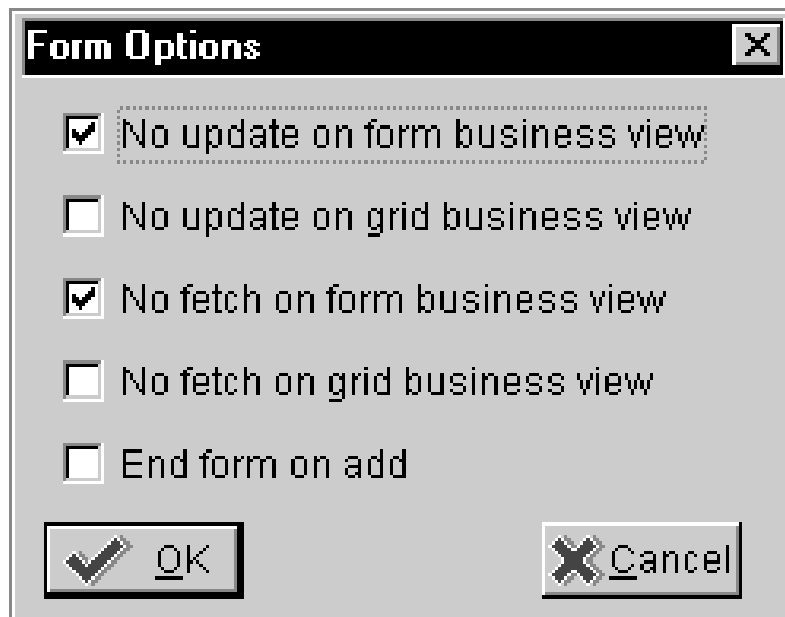
An empty default data structure is created for any report section with an attached business view. This data structure is used to receive values from another report in a

report interconnection. If a required item is not in the existing data structure, you can select 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 same procedures as those for modifying 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. Suppose you create two Fix/Inspect forms that use the same business view. The second form shows more detail, for example, 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 the information (fields on the form)/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 letting the first form update the database. In this case, you should set the form options for updating the database to No Update on Form Business View.



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 this 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 first form's business view. Use a business view for the first form that has only the columns needed for the first form. If you fetch all columns needed for both forms you do only one Select from the database and all the data is passed over the network once, as opposed to having two separate Select statements generated by two

different business views traversing the network. If hardware or memory for the workstation is not a problem, then fetch all columns for that record for both forms and let that information reside in the workstation's memory.

Displaying Type Definitions

You can display the typedef for a data structure.

► To display typedef

1. On Parameter Design choose the data structure.

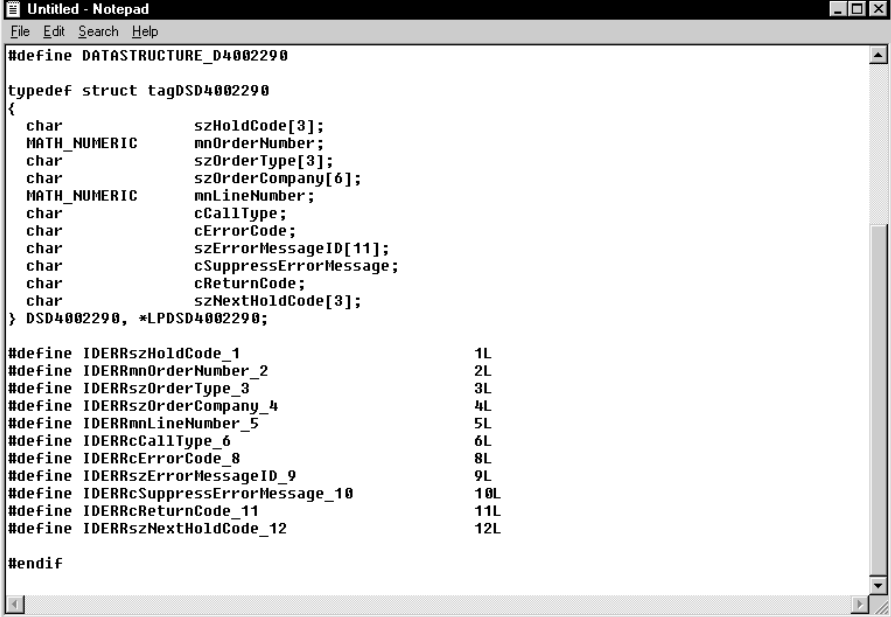
The screenshot shows the 'Object Librarian - [Parameter Design]' window. The 'Template Name' is 'D4002290' and the 'Object Type' is 'DSTR'. The 'Description' is 'F4209 Verify Order For Holds'. The 'System Code' is '42' (Sales Management) and the 'Reporting System Code' is '42' (Sales Management). The 'Object Use' is '330' (General). Below this is a table with columns: Location, Path Code, S C, SAR, User ID, Date Modified, Time Modified, and Modification Comment.

Location	Path Code	S C	SAR	User ID	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1		LP2174240	8/27/96	15:13:48	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3		BS5625464	8/2/98	16:39:09	

Row:3

2. From the Form menu, choose typedef.

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



```

Untitled - Notepad
File Edit Search Help
#define DATASTRUCTURE_D4002290
typedef struct tagDSD4002290
{
    char            szHoldCode[3];
    MATH_NUMERIC    mnOrderNumber;
    char            szOrderType[3];
    char            szOrderCompany[6];
    MATH_NUMERIC    mnLineNumber;
    char            cCallType;
    char            cErrorCode;
    char            szErrorMessageID[11];
    char            cSuppressErrorMessage;
    char            cReturnCode;
    char            szNextHoldCode[3];
} DSD4002290, *LPDSD4002290;

#define IDERRszHoldCode_1          1L
#define IDERRmnOrderNumber_2      2L
#define IDERRszOrderType_3        3L
#define IDERRszOrderCompany_4     4L
#define IDERRmnLineNumber_5       5L
#define IDERRcCallType_6          6L
#define IDERRcErrorCode_8         8L
#define IDERRszErrorMessageID_9   9L
#define IDERRcSuppressErrorMessage_10 10L
#define IDERRcReturnCode_11       11L
#define IDERRszNextHoldCode_12    12L

#endif
    
```

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. OneWorld provides the ability to create an encapsulated data structure object.

You can create several types of data structures that are maintained independently as objects in Object Librarian. You can

- Create a media object data structure
- Create a processing options data structure
- Create a business function data structure

Refer to the *Development Standards: Application Design Guide* for J.D. Edwards naming conventions.

Creating a Media Object Data Structure

A special data structure object is required to work with media objects. The F00165 table 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 entered. The F00165 table contains the following:

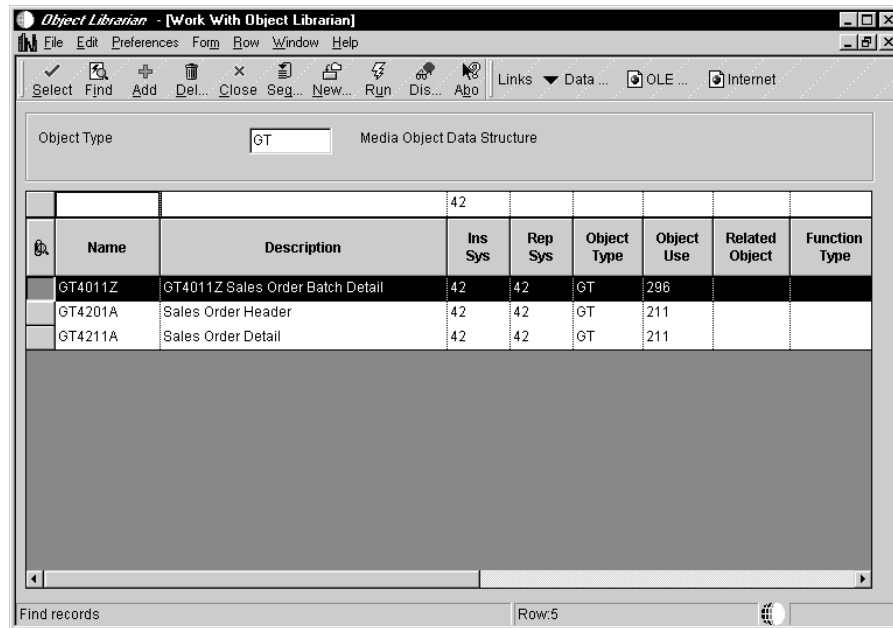
- Object Name (for example GT4201A)
- GDTXKY (Key) (for example, sales order number | order type | company)
- GDTXVC (Media Object Attachment)

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



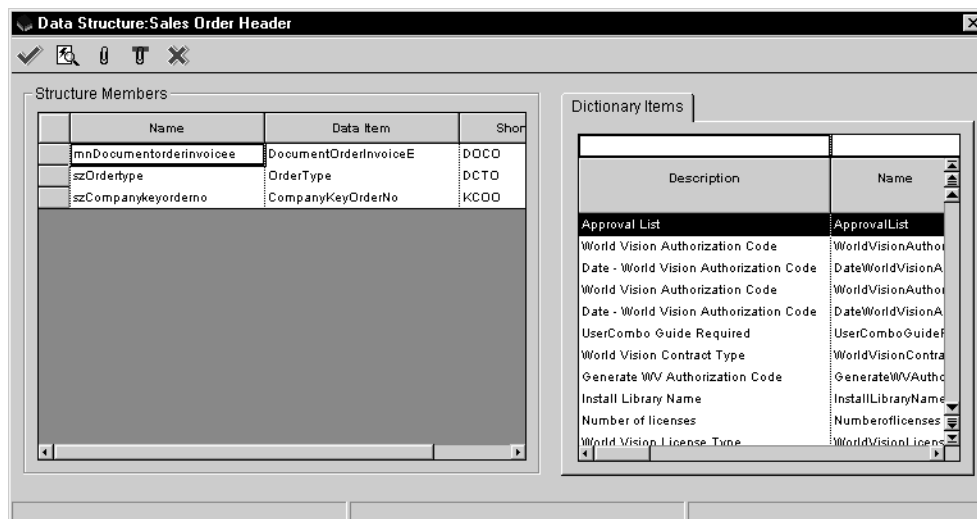
To create a media object data structure

On Object Librarian



1. Add a data structure object (type GT) or choose an existing one to modify.

The J.D. Edwards standard format for naming a media object data structure is GTxxxx.



2. Choose data items from Dictionary Items list.
 - Enter a Name, Alias, System Code or combination in the QBE fields and click Find to more easily locate an item in the list.
3. To change the name double-click on the name and type a new name. Use Hungarian notation rules.

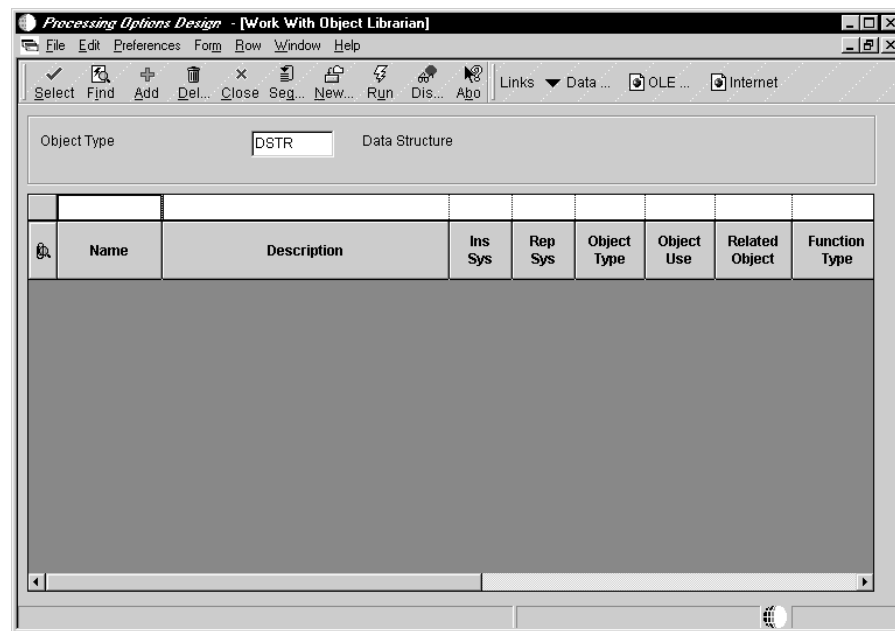
4. Drag the desired data item to Structure Members.
5. Click OK.

Creating a Processing Options Data Structure

Processing options are used to create an input property sheet.

► To create a processing options data structure

From Cross Application Development (GH902), choose Processing Options Design



1. On Work with Object Librarian, complete the following field:
 - Object Type

You must enter DSTR for data structure.

2. Click Add.

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

- Object Name
- Description
- System Code
- Reporting System Code
- Object Use
- Category Code - Object Librarian 06

Make certain Structure Type is 2.

- SAR Number

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 Work with Object Librarian, complete the following field:

- Object Type

You must enter DSTR for data structure.

2. Click Add.

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

- Object Name
- Description
- System Code
- Reporting System Code
- Object Use
- Category Code - Object Librarian 06

Make certain Structure Type is 1.

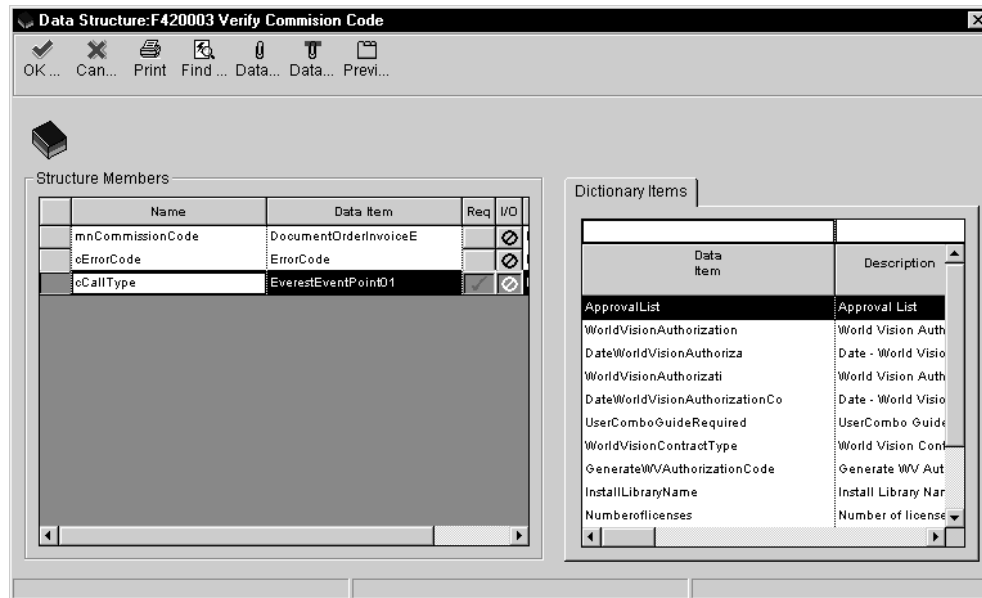
- SAR Number

4. Click OK.

5. From the Object Librarian form choose the data structure you just created.

6. Click the Design button.

The following form appears. You can locate Data Dictionary items on the right side and drag them to the left, adding data items as needed.





Cross Reference Facility

You can use the Cross Reference Facility to find out information about where specific kinds of objects are used and how they are used. You can also view relationships between objects and their components. For example, you can:

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

You can rebuild cross reference relationships if you change something.

This section describes the following:

- ☐ Working with the Cross Reference facility



Working with the Cross Reference Facility

The Cross Reference Facility (P980011), is located on the Cross Application Development Tools menu.

Use Cross Reference for:

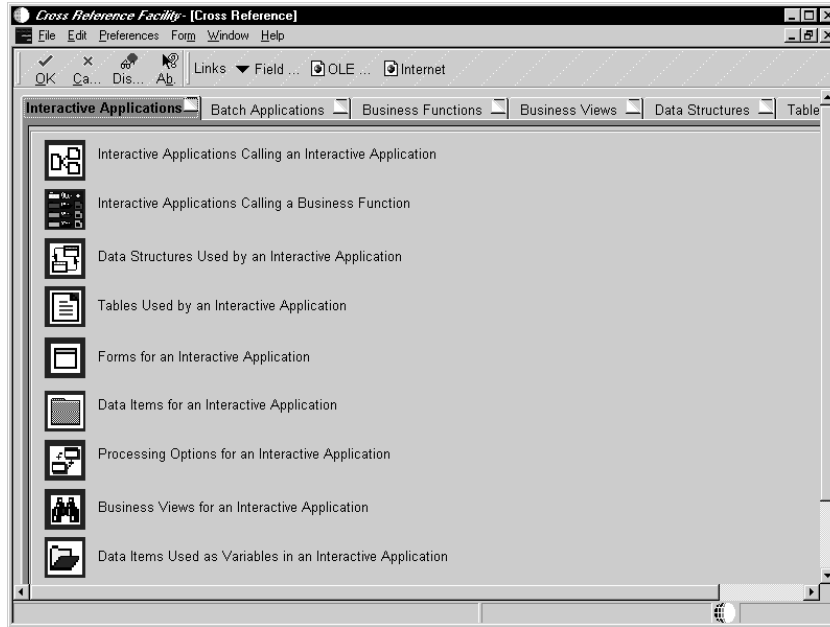
- ☐ Searching for objects
- ☐ Searching for event rules
- ☐ Viewing field relationships
- ☐ Rebuilding cross reference information

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.



Click one of the following tabs:

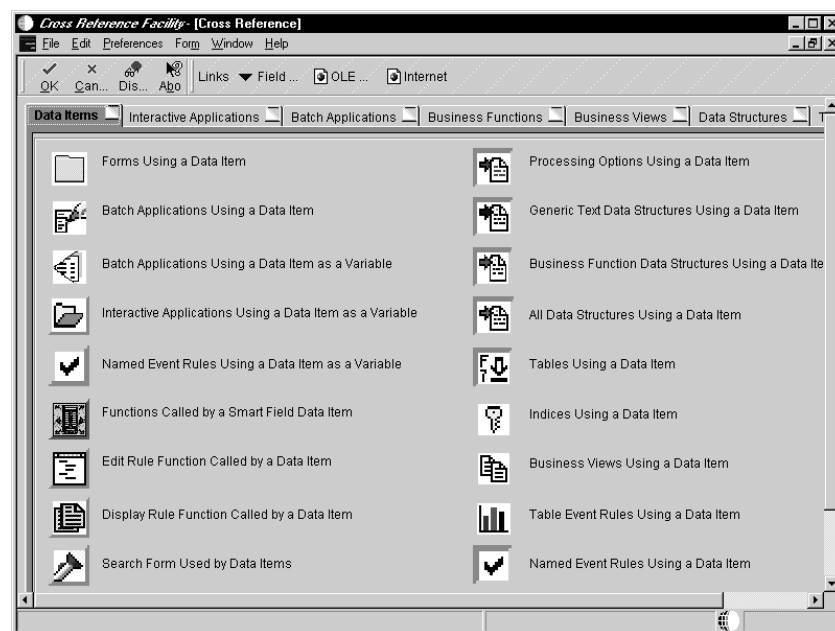
- Data Items
- Interactive Applications
- Batch Applications (the batch application object type in Cross Reference is UBE)
- Business Functions
- Business Views
- Data Structures
- Tables
- Forms

Searching for Data Items

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

- Forms using a data item
- UBEs using a data item
- UBEs event rules using a data item as a variable
- Applications using a data item as a variable
- Named event rules using 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 using a data item
- Generic text data structures using a data item
- Business function data structures using a data item
- All data structures using a data item
- Tables using a data item
- Indices using a data item
- Business views using a data item
- Table event rules using a data item

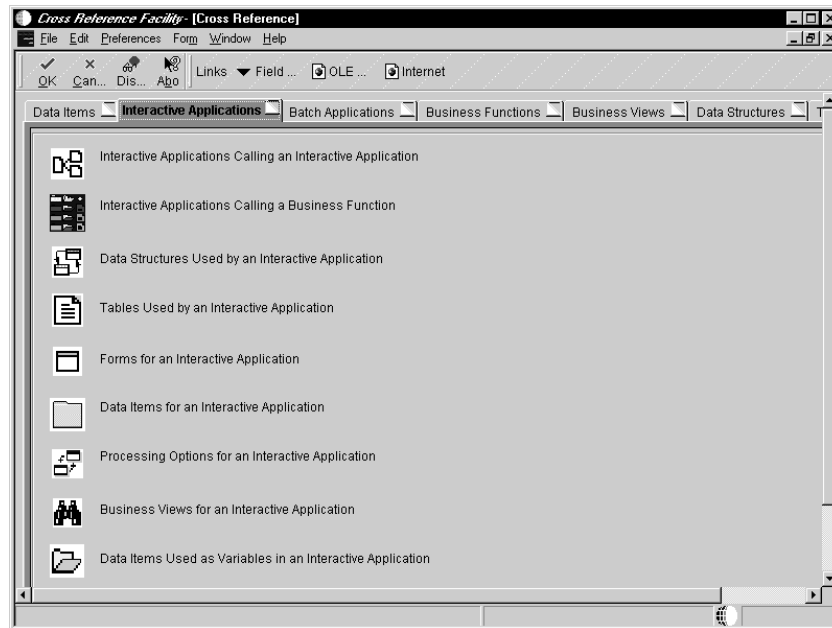


Searching for Interactive Applications

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

- Applications calling an application
- Applications calling a business function
- Data structures used by an application
- Tables used by an application
- Forms for an application
- Data items for an application

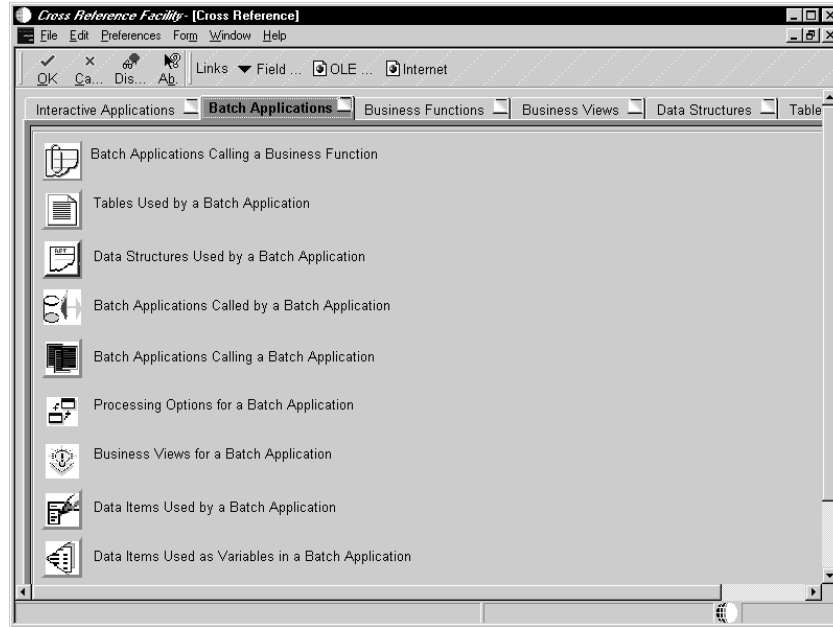
- Processing options for an application
- Business views for an application
- Data items used as a variable in an application



Searching for Batch Applications

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

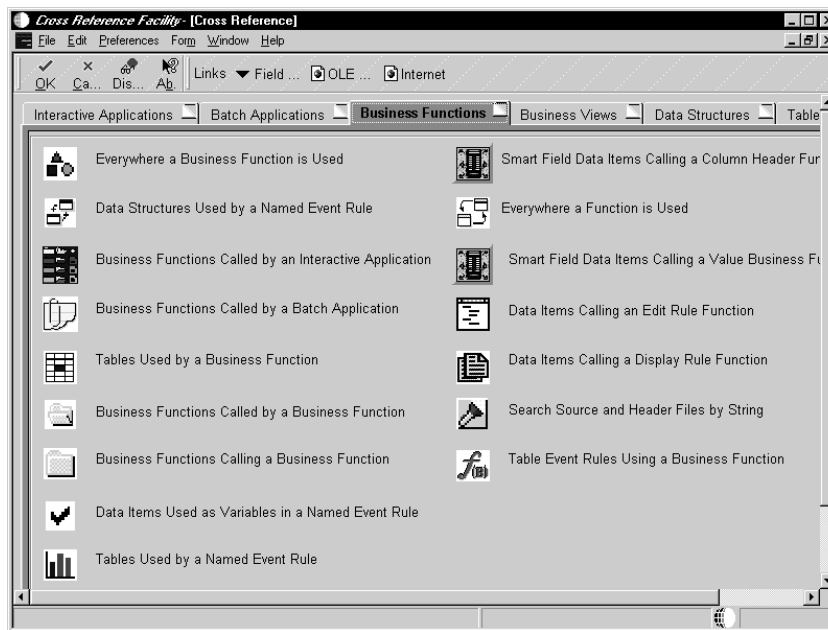
- UBEs calling a business function
- Tables used by a UBE
- Data structures used by a UBE
- UBEs called by a UBE
- UBEs calling a UBE
- Processing options for a UBE
- Business views for a UBE
- Data items used by a UBE
- Data items used as variables in a UBE



Searching for Business Functions

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

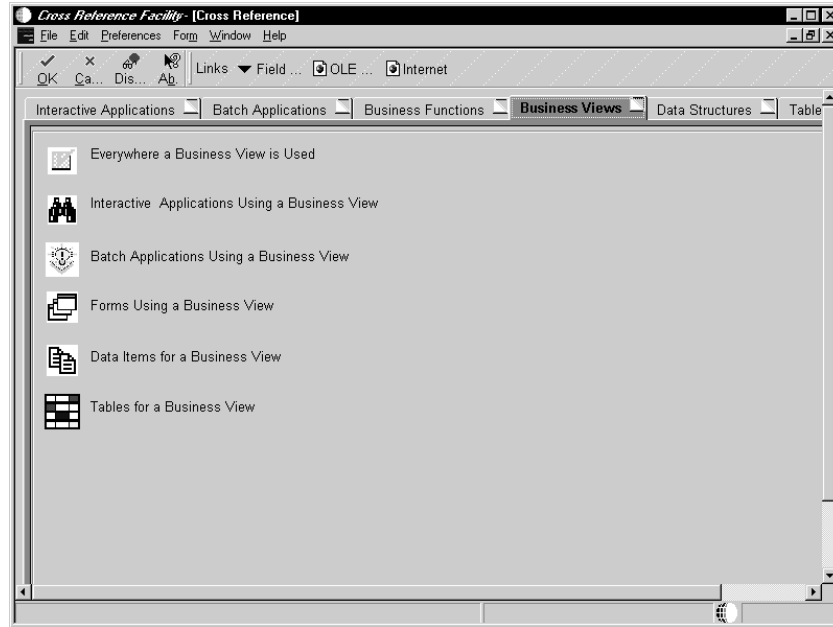
- Everywhere a business function is used
- Data structures used by a named event rule
- Business functions called by an application
- Business functions called by a UBE
- Tables used by a business function
- Business functions called by a business function
- Business functions calling a business function
- Data items used as variables in a named event rule
- Tables used by a named event rule
- Business functions called by a named event rule
- Smart field data items calling a column header function
- Everywhere a function is used
- Smart field data items calling a value business function
- Data items calling edit rule functions
- Data items calling display rule functions
- Search source and header files by string
- Table event rules using 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:

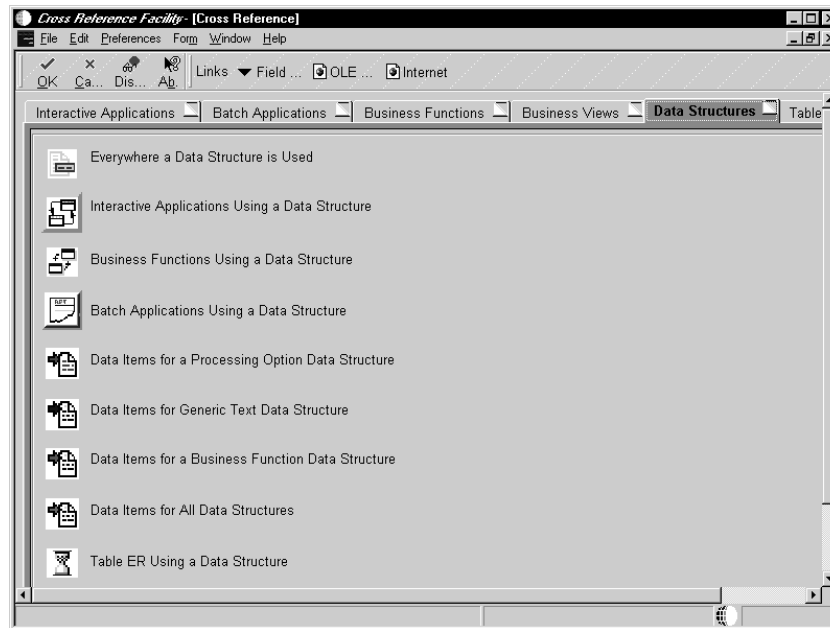
- Everywhere a business view is used
- Applications using a business view
- UBEs using a business view
- Forms using a business view
- Data items for a business view
- Tables for a business view



Searching for Data Structures

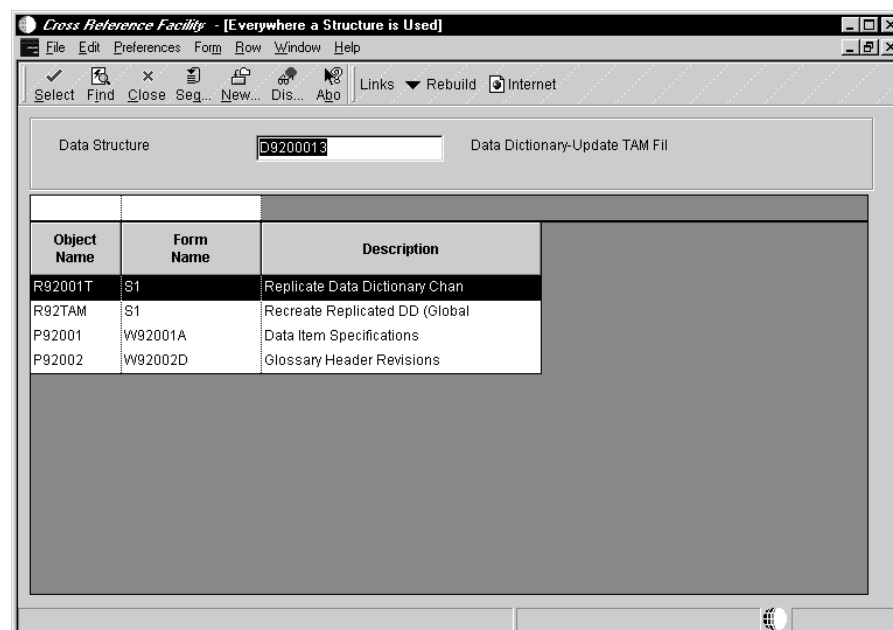
You can locate information about data structures and how they are used. You can search for the following:

- Everywhere a data structure is used
- Applications using a data structure
- Business functions using a data structure
- UBEs using 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 using a data structure



Everywhere a Data Structure is Used

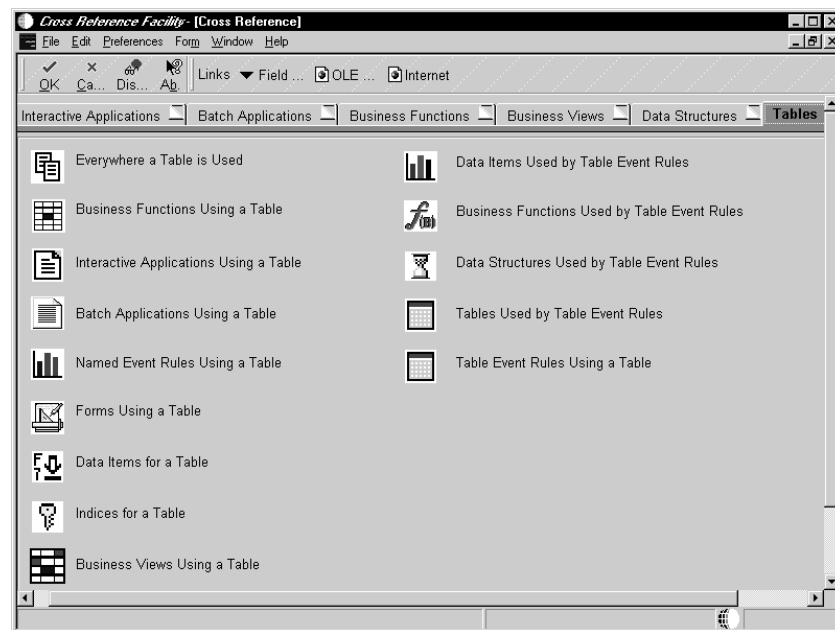
You can determine all the places that a specific data structure is used.



Searching for Tables

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

- Every a table is used
- Business functions using a table
- Applications using a table
- UBEs using a table
- Named event rules using a table
- Forms using a table
- Data items for a table
- Indices for a table
- Business views using a table
- Data items used by table event rules
- Business functions used by table event rules
- Data structures used by table event rules
- Tables used by table event rules
- Table event rules using a table

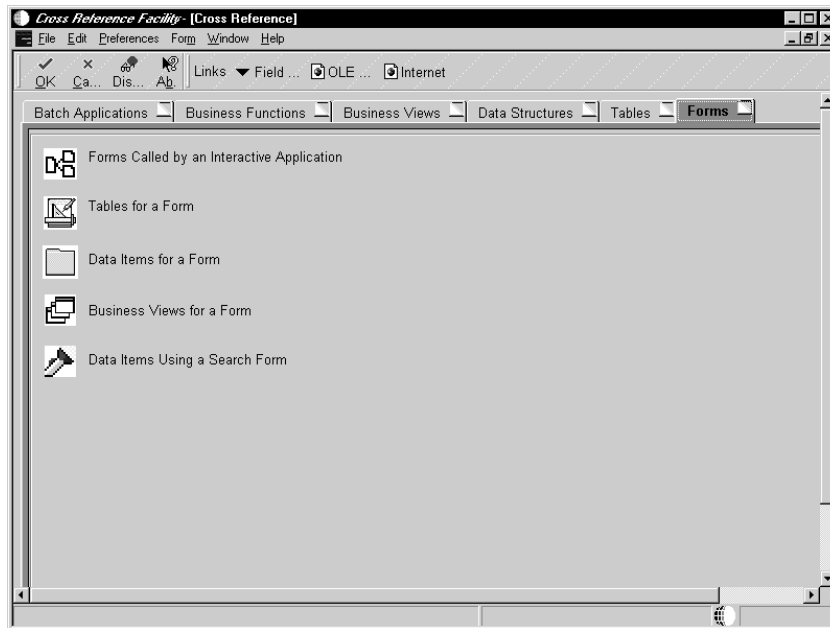


Searching for Forms

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

- Forms called by an application
- Tables for a form

- Data items for a form
- Business views for a form
- Data items using a search form



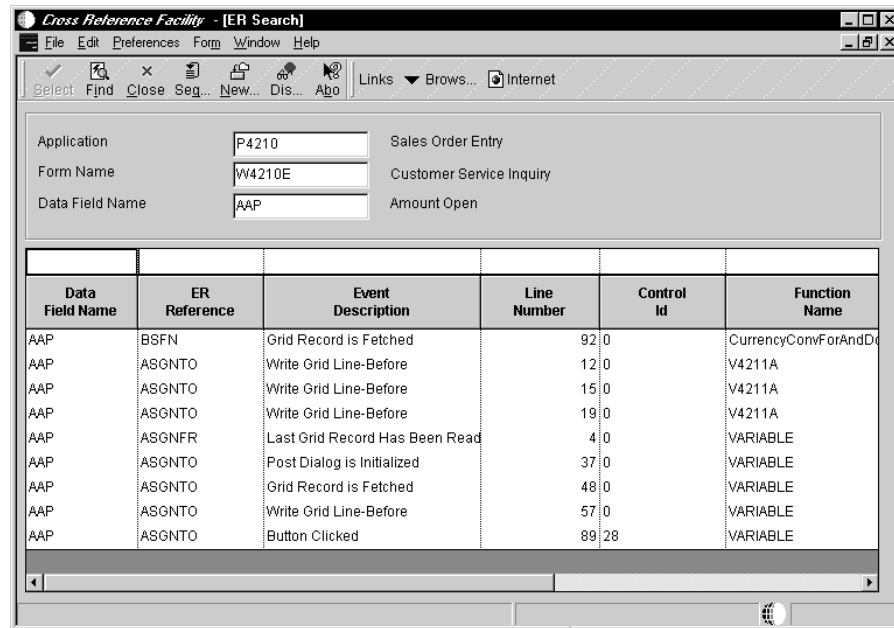
Searching for Event Rules

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

► To search for event rules

On the search form you have chosen

1. From the Form menu choose Event Rules.



2. Complete the following fields

- Application
- Form Name
- Data Field Name

Field	Explanation
Application	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Data Field Name	The Data Field Name field simply provides the name of a data field being used in either a data base file, a form or a report. This field name is the six character RPG name being referenced in the Data Description Specifications.
Form Name	The name given to a record format for a form, report, or database table.

You can also choose BrowsER from the form menu to browse through event rules. Refer to BrowsER in this guide for more information.

Viewing Field Relationships

The Field Relationships form is meaningful only for the following Cross Reference search types:

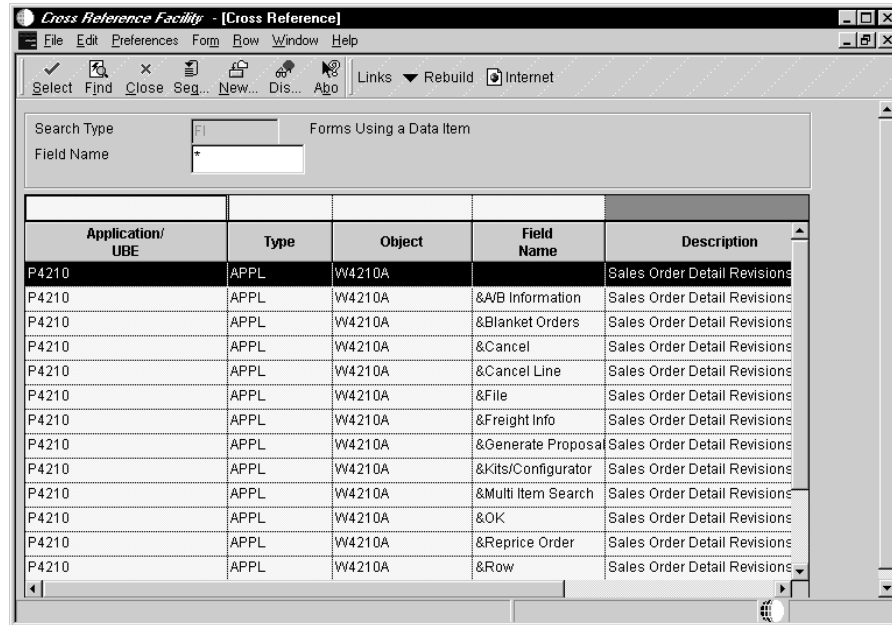
DA	Data items used by an application
FA	Forms for an application
FI	Forms using a data item
SA	Data structure for an application

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

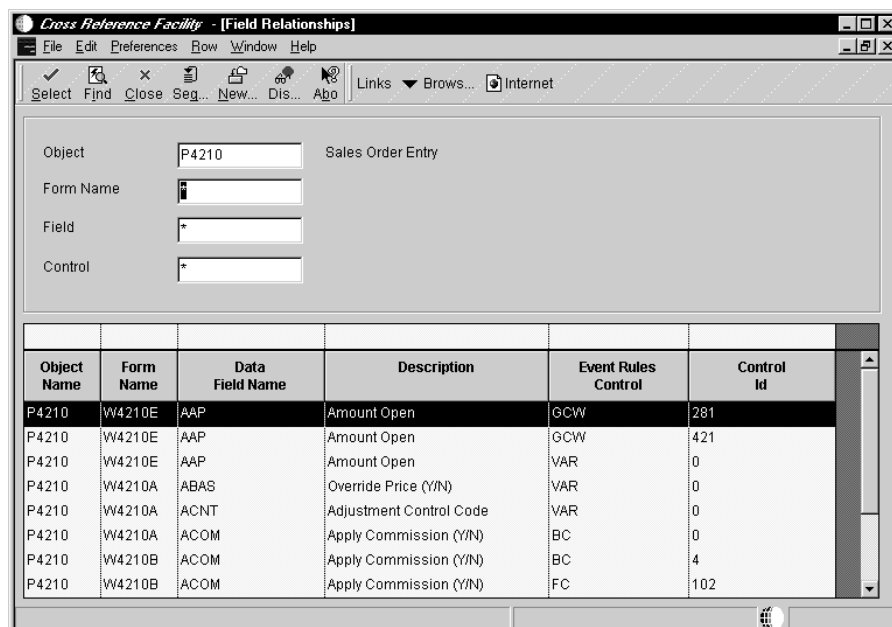
- BC for a business view column
- FI for a form interconnect control
- GC for a grid control
- FC for a form control

To view field relationships

On the Cross Reference search form you have chosen



Choose Relationships from the Form menu.



Rebuilding Cross Reference Information

Whenever objects are modified, cross reference information is out of sync with objects in the system. Because the cross reference files are not automatically rebuilt when objects are modified, it is necessary to do so at the time of inquiry. You may also do regularly scheduled builds to ensure that the cross reference information is put in sync on a regular basis.

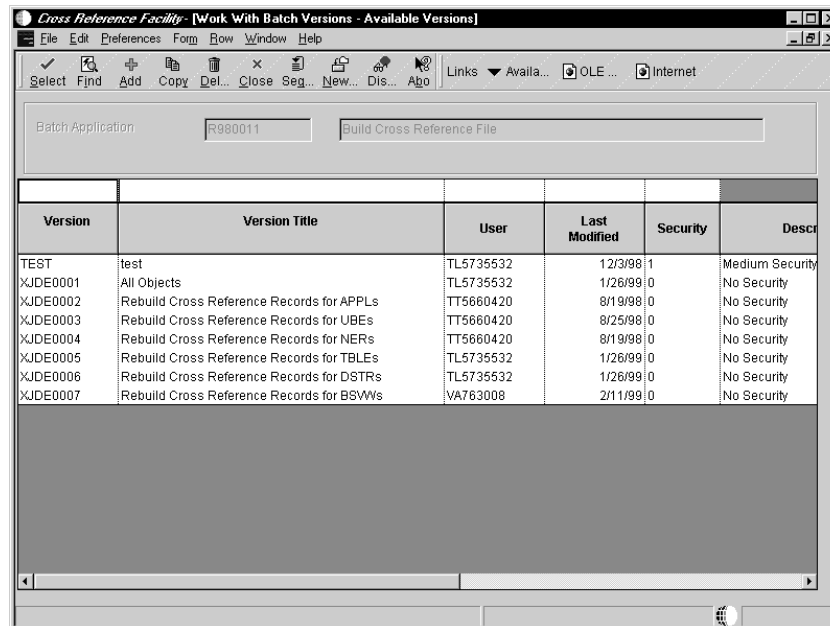
View the date a record was built in the far right column of the grid on each Cross Reference form of the Cross Reference utility. If information is out of date, use the Rebuild option from any of the Cross Reference forms.

Cross Reference builds using relational database tables, not local specifications.

► To rebuild cross reference information

On Cross Reference

1. From the Form menu choose Rebuild.



2. Choose the objects you want to rebuild.

This process can take several minutes.



Checkout Log

The Object Librarian logs all checkin, checkout, and delete operations to a Checkout Log. This log is stored in the system data source in the Checkout Log table - F9882. Checkout Log logs all check-ins, check-outs, and transfers of objects. A transfer is logged as a check-in. The Work with Checkout Log Application (P9882) allows developers and version control personnel to view the Checkout Log when it has been moved into the central objects data source. Use the Checkout Log to:

- Confirm that an object has been checked in
- Run the Checkout Log report
- Purge the Checkout Log

To access these functions go to the Object Management menu, GH9081.

This section describes the following:

- ☐ Working with the Checkout Log



Working with the Checkout Log

After an object has been moved to the central objects database, you can track the status of the object by using the Checkout Log. You can determine when an object has been checked out or checked in. You can also delete or install an object using the Checkout Log Application.

This chapter describes the following:

- ☐ Confirming that an object has been checked in
- ☐ Purging the Checkout Log

Confirming That an Object Has Been Checked In

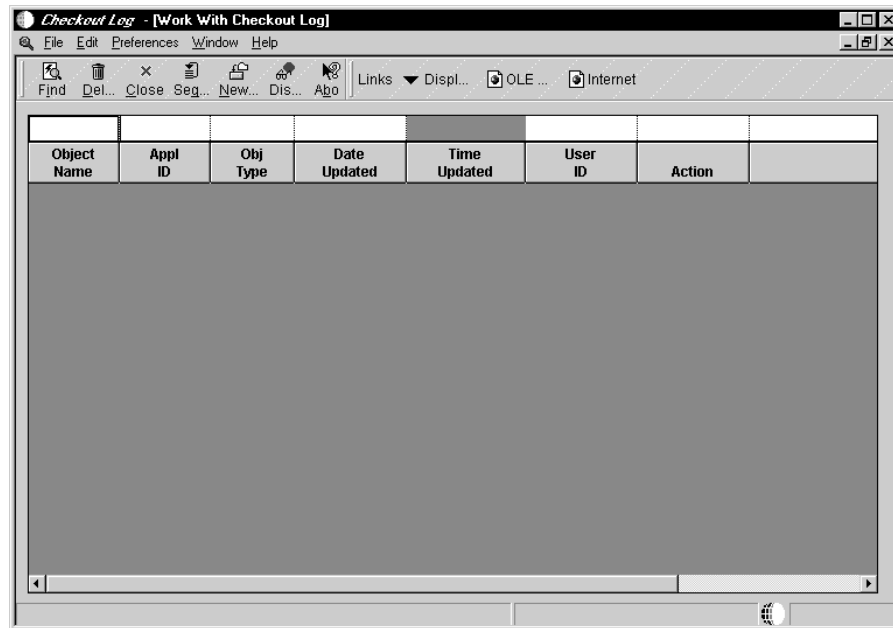
An object might have been checked out for enhancements. To ensure that it has been checked back into the central database, or to review its status, use the Checkout Log.



To confirm that an object has been checked in

On Object Management (GH9081)

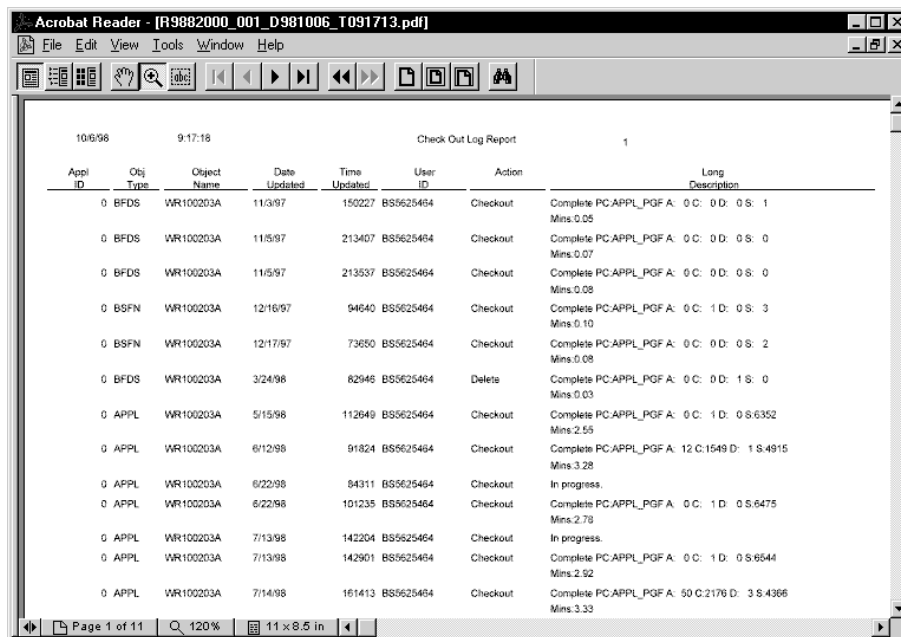
1. Double-click on Checkout Log.
2. On Checkout Log, double-click on Checkout Log.
3. Enter an object name and click Find.



Field	Explanation
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Application ID - OneWorld	The Application ID uniquely identifies the OneWorld Application.
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.
Date - Updated	The date of the last update to the file record.
Object Check In Time	The time the object was last checked in.
User ID	For World, the IBM-defined user profile. For OneWorld, the identification code for a user profile.
Description	A user defined name or remark.

Purging the Checkout Log

The Checkout Log file grows by many records daily. Use the checkout log purge to delete outdated logs to ensure that the log is current and to keep it a reasonable and useful size. The purge can be executed as a batch job upon exiting an application. The batch process should be based on the Date Updated field in the Checkout Log table. Remove old records from the table on a regular basis.



Appl ID	Obj Type	Object Name	Date Updated	Time Updated	User ID	Action	Log Description
0	BDFS	WR100203A	11/3/97	150227	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 0 D: 0 S: 1 Mins.0.05
0	BDFS	WR100203A	11/5/97	213407	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 0 D: 0 S: 0 Mins.0.07
0	BDFS	WR100203A	11/5/97	213537	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 0 D: 0 S: 0 Mins.0.08
0	BSFN	WR100203A	12/16/97	94640	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 1 D: 0 S: 3 Mins.0.10
0	BSFN	WR100203A	12/17/97	73650	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 0 D: 0 S: 2 Mins.0.08
0	BDFS	WR100203A	3/24/98	82946	BS5625464	Delete	Complete PC:APPL_PGF A: 0 C: 0 D: 1 S: 0 Mins.0.03
0	APPL	WR100203A	5/15/98	112649	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 1 D: 0 S: 6352 Mins.3.55
0	APPL	WR100203A	6/12/98	91824	BS5625464	Checkout	Complete PC:APPL_PGF A: 12 C: 1549 D: 1 S: 4915 Mins.3.28
0	APPL	WR100203A	6/22/98	84311	BS5625464	Checkout	In progress.
0	APPL	WR100203A	6/22/98	101235	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 1 D: 0 S: 6475 Mins.2.78
0	APPL	WR100203A	7/13/98	142204	BS5625464	Checkout	In progress.
0	APPL	WR100203A	7/13/98	142901	BS5625464	Checkout	Complete PC:APPL_PGF A: 0 C: 1 D: 0 S: 6544 Mins.2.62
0	APPL	WR100203A	7/14/98	161413	BS5625464	Checkout	Complete PC:APPL_PGF A: 50 C: 2176 D: 3 S: 4366 Mins.3.33



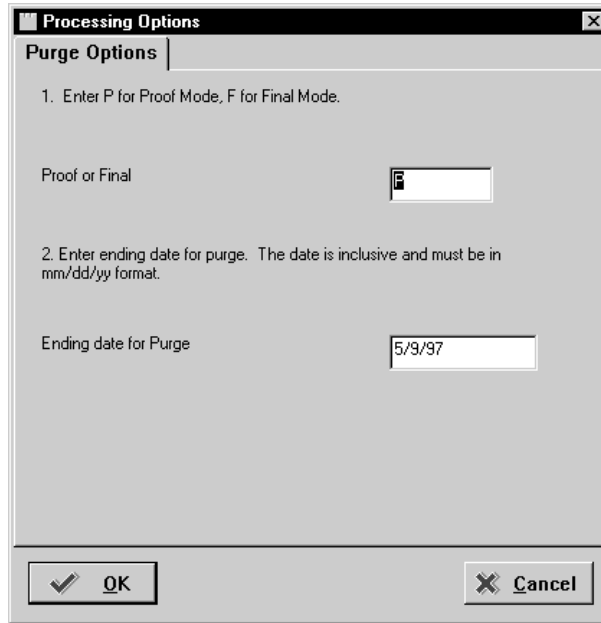
Not all users can purge the Checkout Log. The access rights to purge the checkout log are determined by your system administrator.



To purge the Checkout Log

From the Checkout Log menu

1. Double-click on Purge Checkout Log (R9882000, version 002).



The image shows a Windows-style dialog box titled "Processing Options" with a sub-tab labeled "Purge Options". The dialog contains two numbered instructions: "1. Enter P for Proof Mode, F for Final Mode." and "2. Enter ending date for purge. The date is inclusive and must be in mm/dd/yy format." Below the first instruction is a text input field containing the letter "P". Below the second instruction is a text input field containing the date "5/9/97". At the bottom of the dialog are two buttons: "OK" with a checkmark icon and "Cancel" with an "X" icon.

Processing Options

Purge Options

1. Enter P for Proof Mode, F for Final Mode.

Proof or Final

2. Enter ending date for purge. The date is inclusive and must be in mm/dd/yy format.

Ending date for Purge

☒ OK ☐ Cancel

You can use one of the following options:

- To see a detailed listing of the checkin checkout process run a version of the R9882000 batch program. Create your own version by copying one of the supplied versions and define custom data selection and processing options for your version. By using data selection you can narrow down what you want to see, such as by UserID or by Object Name and so forth. Set the processing options so option 1 will be a "p" for Proof mode, and enter the appropriate date for option 2.
- To purge the F9882, run a new version of the same batch process (R9882000). Create your custom version, which will have your custom data selection, for items to purge by UserID or Object Name, and processing options. For this version, for processing option enter an F, and for processing option 2 enter the appropriate date.

Application Design



Application Design

Application Design is the entry point to several tools for creating, generating, running, maintaining, and securing applications. Application Design includes Forms Design for creating forms and Event Rules Design for attaching business logic through event rules. Use Application Design to:

- Access Forms Design for creating forms
- Define processing options
- Run an application
- Access BrowsER
- Create text overrides
- Browse forms in an application

You can use the OneWorld toolset to create applications in different modes, including:

- Windows client
- Java client
- HTML client

See Also

- *Forms Design* for complete instructions on creating a form
- *Processing Options* for creating and attaching a processing options template to an application
- *BrowsER* for instructions on viewing, enabling, and disabling event rules used within forms
- *Developing Web Applications* for more information about developing Web applications.

This section describes the following:

- ☐ Understanding Application Design
- ☐ Adding an interactive application



Understanding Application Design

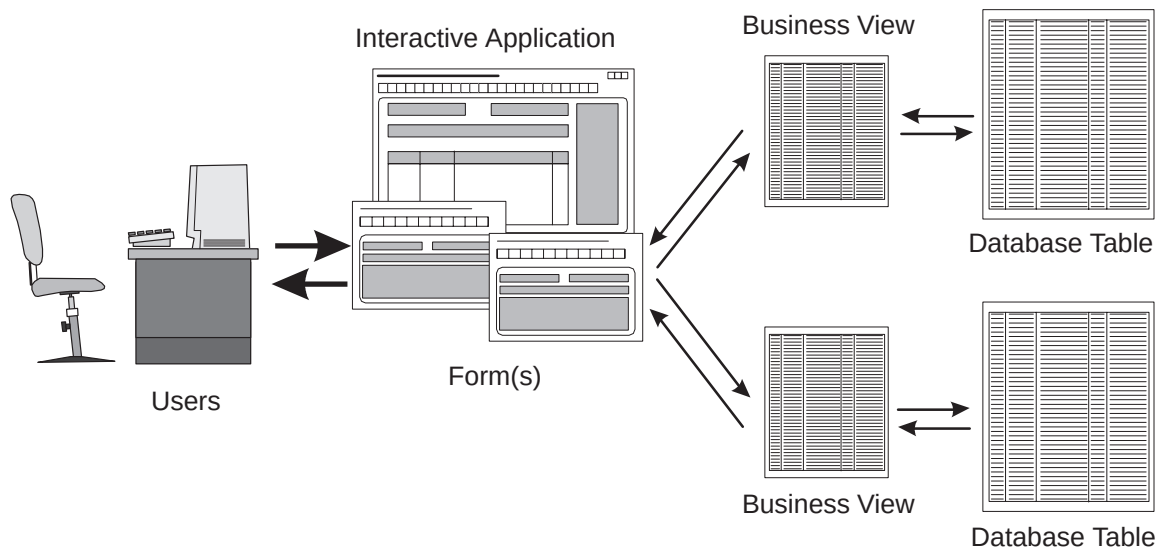
An application is a collection of computer programs that perform a specific task. An application retrieves and updates data within a database table. Accounts Payable and Sales Order Processing are examples of applications.

In OneWorld, there are two types of applications:

- Interactive applications
- Batch applications

Understanding Interactive Applications

An interactive application is the interface between a user and a database table (based on a business view). A user uses an interactive application to add, modify, or view data using a form.



Understanding Batch Applications

A batch process is an application that processes automatically without user interaction. Table conversions and reports are examples of batch processes. See *Batch Processing* for detailed information about batch processes.

Understanding Web Applications

To create Web applications, use the OneWorld toolset in the same manner as you would for standard applications. You can then customize your windows applications for the Web.

In addition to installing the Web server, you may need to install the Java generator on the OneWorld developer workstation. Install the Java generator if you are planning to develop new applications or change existing applications. The Java Generator is usually automatically installed when you do a OneWorld client install.

Refer to *Developing Web Applications* in this guide for more information about developing web applications.

Refer to the *Web-Based Solutions* guide for more information about Web products.

Adding an Interactive Application

An application is an object. You must add an application so it exists as an object before you can begin developing it.

► To add an interactive application

On Work with Object Librarian

1. Enter APPL in the Object Type field.
2. Click Add.

The screenshot shows the 'Object Librarian - [Add Object]' dialog box. It has a menu bar with 'File', 'Edit', 'Preferences', 'Window', and 'Help'. Below the menu bar is a toolbar with icons for 'OK', 'Cancel', 'Dis...', 'Add', 'Links', 'Displ...', 'OLE...', and 'Internet'. The main area of the dialog contains several input fields and labels:

- Object Name:** P42001
- Object Type:** APPL
- Description:** Sales Order Processing Demo2
- System Code:** 42 (Sales Management)
- Reporting System Code:** 42 (Sales Management)
- Object Use:** Undefined
- SAR Number:** (empty field)

3. On Add Object, complete the following required fields:
 - Object Name
 - Description
 - System Code

Refer to the *J. D. Edwards Naming Standards* below.

4. Complete the following optional fields:

- Reporting System Code
- Object Use
- SAR Number

5. Click OK.

After the application exists as an object, you can start building the components of the application. You can use Forms Design to design the first form in your application. Instructions for using Forms Design are contained in the next section.

J.D. Edwards Naming Standards

Use the following guidelines when you add an application:

The Object Librarian name for an application can be a maximum of 8 characters formatted as follows: **Pxxxxyyy**

The name field accepts up to 10 characters. However, if you enter more than 8 characters, the entry will be truncated.

P = application

xxxx = the system code

yyy = a next number, such as 001 and 002

Provide up to a 60-character description. It should reflect the subject of the forms within the application, such as Companies and Constants.



Forms Design

Use Forms Design to create one or more forms for an application.

This section describes the following

- ☐ Understanding form types
- ☐ Creating a form
- ☐ Designing a form layout
- ☐ Working with menu/toolbar exits
- ☐ Working with controls
- ☐ Using text variables
- ☐ Using Quick Form
- ☐ Processing media objects
- ☐ Testing a form

Before You Begin

Before working with Forms Design, you should have a working knowledge of Windows and have an understanding of the OneWorld application design process. Additionally, you must:

- ☐ Think about which data items are required for each form in the application.
- ☐ Determine if there are any existing tables that suit your needs. If none exist, create a table.
- ☐ Create a business view on which to base a form.

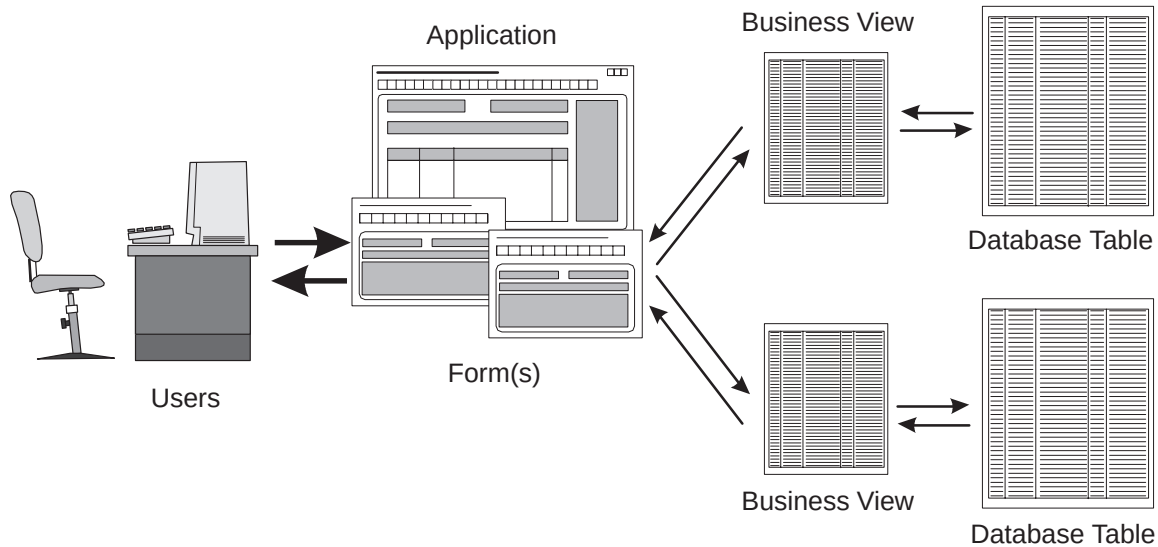
See *About Table Design* and *About Business View Design* in this guide for information about creating tables and business views.



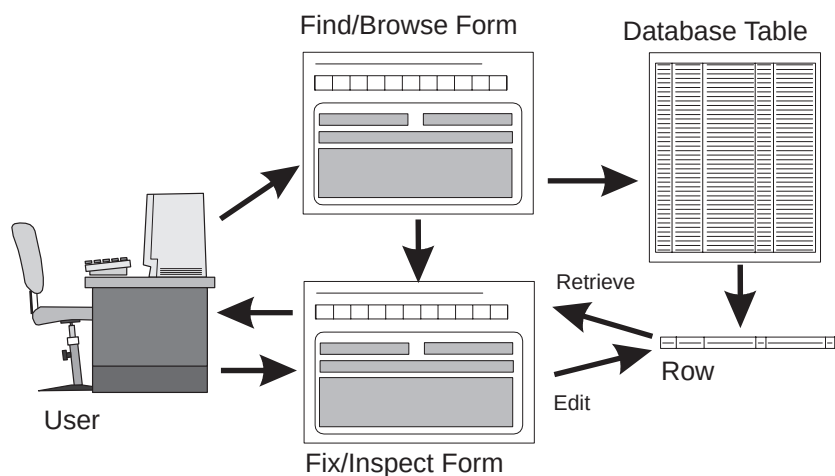
What is a Form?

A form is the interface between a user and a table. This interface should:

- Present data logically
- Contain the functionality necessary to enter and manipulate data



A single application can contain one or more forms. Usually, a Find/Browse form is the first form displayed in the application. It enables the user to locate a specific record with which he or she wants to work. Upon selecting a record, a subsequent form, such as a Fix/Inspect form, is automatically displayed, where the user can view or modify data for that record.



Elements of a Form

Form Type	The form type establishes the basic functionality of a form. Each form type has default controls and processes.
Business Views	In an application, business views link forms and tables. You must associate all forms except the Message form with a business view.
Controls	All objects on a form are controls. Controls include grids, check boxes, radio buttons, push buttons, and more.
Properties	In application design, there are several types of properties: application, form, control, and grid. Properties define appearance and functionality.
Data Structure	A data structure defines the data that can be passed between forms within an application or between applications. Once the data structure of a form is defined, use form interconnections to indicate the direction of flow of data between forms.
Event Rules	Event Rules can contain processing instructions for specific events. Events are actions that occur on a form, such as clicking a button or tabbing out of a field. Use event rules to attach business logic to any event. Events are triggered either as a result from user interaction with a control (for example, click button) or as a result of a system controlled process (for example, loading a grid).

See *Understanding Form Types* for a discussion of each form type.

Understanding Form Types

When you create a new form, you must initially specify a form type. It is important to understand the available form types, because each form has characteristics that accommodate different tasks. For example, the Find/Browse form includes a Find option to search for records across the table and a grid to display those records.

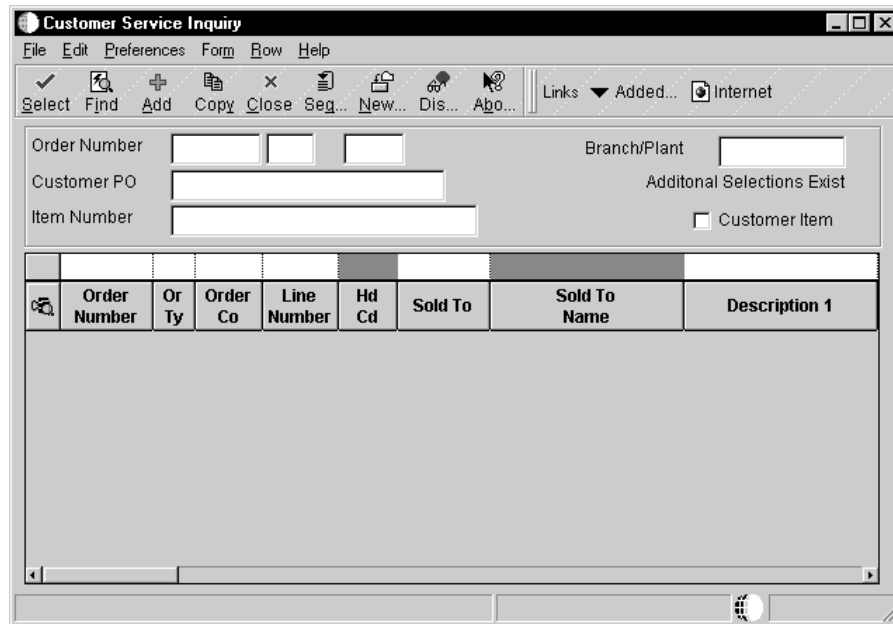
The OneWorld Forms Design tool supplies the following form types:

- Find/Browse
- Parent/Child
- Fix/Inspect
- Header Detail
- Headerless Detail
- Search and Select
- Message Form

About J.D. Edwards Design Standards

J.D. Edwards design standards ensure a consistent approach across all applications. While many of these standards apply to all form types, there are also specific standards for each form type. See the *Development Standards: Application Design Guide* for a complete list of standards.

Find/Browse Forms



Find/Browse is the entry point to an application. It contains an optional query-by-example (QBE) line so you can search on any database field in the grid. QBE columns that are grayed out and do not have QBE capability (for example, Sold To Name in the example above).

Use a Find/Browse form to:

- Search, view, and select multiple records in a grid
- Delete records
- Exit to another form to add, update, or view a table record

Find fetches records; the filter fields and QBE line provide the WHERE clause of a SELECT statement. Delete removes records from the table.

You cannot add new or update existing records on a Find/Browse form.

A Find/Browse form displays data contained in one table. Therefore, you can attach only one business view to a Find/Browse form.

The following table describes toolbar functionality for this form.

Select

Standard toolbar button that comes with the form. You must add the appropriate form interconnections to provide functionality.

Find	Standard toolbar button that comes with the form
Close	Standard toolbar button that comes with the form
Add	You can add this button to the toolbar. You must also provide the functionality for the button, which typically includes a form interconnect.
Copy	You can add this button to the toolbar. You must also provide the functionality for the button.
Delete	You can add this button to the toolbar. You must also provide the functionality for the button.
OK	This button does not apply to this form type.
Cancel	This button does not apply to this form type.

Fix/Inspect Forms

The screenshot shows a software window titled "Sales Order Header". It features a menu bar with "File", "Edit", "Preferences", "Form", and "Help". Below the menu is a toolbar with icons for "OK", "Can...", "Dis...", and "Ab...", along with a "Links" dropdown menu showing "Sold T..." and "Internet". The form is divided into several sections: "Order Number" and "Previous Order" with input fields; "Attachments" and "Branch/Plant" with a dropdown and a button; a "Foreign" checkbox; "Messages" and "Previous Order Placed On" with a date field; "Address Numbers" with "Sold To" and "Ship To" fields; "Customer PO", "Print Message", "Hold Code", "Tax Area/Code", and "Tax Certificate No." with various input fields; "Dates" with "Order", "Requested", "Scheduled Pick", and "Cancel" fields; "Price Pickslip" with a checkbox; "Invoice Copies" with a field; "Ordered By" with a field; and "Taken By" with a field. The bottom of the window has a status bar with a globe icon.

The Fix/Inspect form allows you to add a new record to a table or to update an existing record. The Fix/Inspect form includes OK and Cancel buttons. When you click OK, updates or additions are written to the table. When you click Cancel, any changes you have made are lost and no database changes are made. Because the Fix/Inspect form only allows you to add or update one record at a time, the form does not contain a grid.

Use this form type to:

- View a single record per form
- Add new records
- Update existing records

Because the Fix/Inspect form only contains one record, you can attach only one business view to a Fix/Inspect form.

If a record was selected on a previous form, the Fix/Inspect form displays data for that record. If no record was selected on the previous form, the Fix/Inspect form is empty, except for any default values.

The following table describes toolbar functionality for this form.

OK	Standard toolbar button that comes with the form.
Cancel	Standard toolbar button that comes with the form.
Select	This button does not apply to this form type.
Find	This button does not apply to this form type.
Close	This button does not apply to this form type.
Copy	This button does not apply to this form type.
Delete	This button does not apply to this form type.
Add	This button does not apply to this form type.

Header Detail Forms

Item Number	Quantity Ordered	Tr. UoM	Unit Cost	Extended Cost	Pu. UoM	Ln Ty	Description
				0.00			

The Header Detail form allows you to work with data from two separate tables. You can use this form to add or update a single header record. You can also add, update, or delete multiple detail records from the same form.

The Header Detail form includes an input-capable grid so you can add or update detail records. Click OK to perform updates or adds to both tables on the form. When you click Cancel, any changes are lost and no database changes are made.

Because the Header Detail form allows you to update or add records from two different tables, you can attach two business views to a Header Detail form. Attach one business view to the grid and the other to the form, updating both tables from a single form. You can use the Header Detail form for one to many relationships.

The following table describes toolbar functionality for this form.

OK	Standard toolbar button that comes with the form.
Cancel	Standard toolbar button that comes with the form.
Find	You can add this button to the toolbar. You must also provide the functionality for the button. The engine will perform a standard fetch just like it does when the form was entered.

Delete	You can add this button to the toolbar. Even though the tool will delete the record, you may wish to add other logic here.
Select	This button does not apply to this form type.
Close	This button does not apply to this form type.
Copy	This button does not apply to this form type.
Add	This button does not apply to this form type.

Headerless Detail Forms

Prefix	Phone Number	Phone Type	Line Number
			1

The Headerless Detail form is used to display multiple records from a single table that is not normalized. Because this form is used to update only one table, you can attach only one business view to the form.

The Headerless Detail form contains an input-capable grid, where you can add or update detail information. The header portion of the form displays data common to all the detail records in the grid. Both header and detail information come from the same business view.

Click OK to perform updates or additions to the table. When the user clicks cancel, any changes you have made are lost, and no database changes are made.

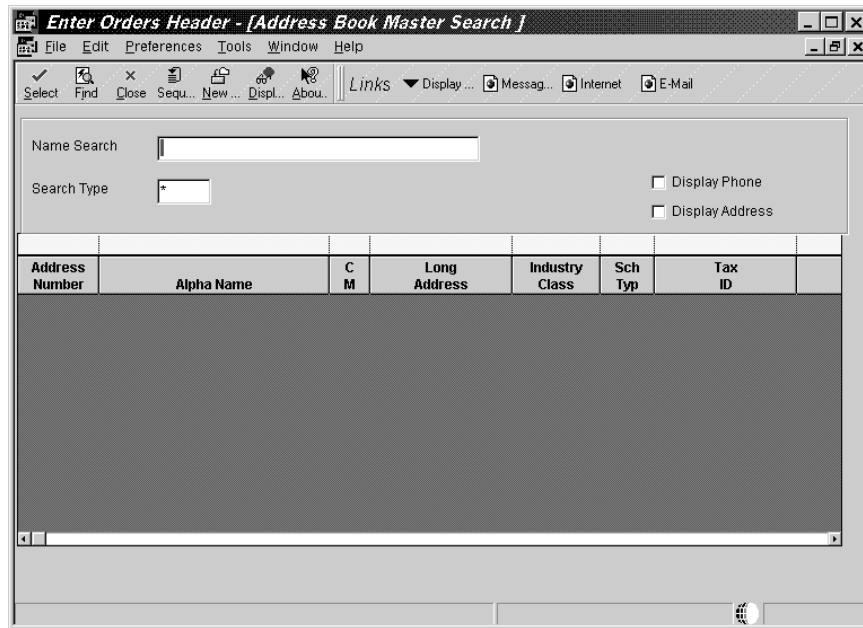
Use this form type to:

- Display multiple records in a grid
- View, add, change, and delete records

The following table describes toolbar functionality for this form.

OK	Standard toolbar button that comes with the form.
Cancel	Standard toolbar button that comes with the form.
Find	You can add this button to the toolbar. You must also provide the functionality for the button. The engine will perform a standard fetch just like it does when the form was entered.
Delete	You can add this button to the toolbar. You must also provide the functionality for the button.
Select	This button does not apply to this form type.
Close	This button does not apply to this form type.
Copy	This button does not apply to this form type.
Add	This button does not apply to this form type.

Search and Select Forms



Use this form to locate a value and return it to the calling field. The Search and Select form is called using a visual assist (flashlight) or hyper-control.

This form only displays information; you cannot edit fields. Therefore, the form contains Select and Close options.

The Search and Select form includes a non-input capable grid where you can view multiple records in one table. The grid displays valid values. When a user chooses a value from the grid and clicks Select, that value is automatically returned to the calling field.

Because this form is used to view records from only one table, you can attach only one business view to a Search and Select form. You use a separate application for this form. One of the benefits of this form is that it improves performance by fetching only necessary fields and it exists in its own application.

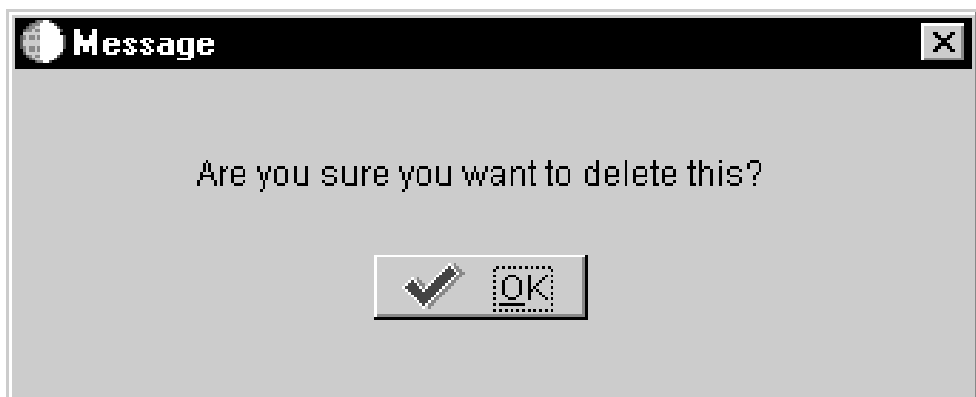
The data structure for this form should contain only one element.

After you create a Search and Select application, you must attach the Search and Select form to the specific data item for which it was created. You do this using the visual assist trigger in the data dictionary or the overrides in the property sheet for a particular control. You use the data dictionary for all instances and override for just one.

The following table describes toolbar functionality for this form.

Select	Standard toolbar button that comes with the form. The Select action will automatically return to the calling form. You do not need to add form interconnects.
Find	Standard toolbar button that comes with the form.
Close	Standard toolbar button that comes with the form.
Copy	This button does not apply to this form type.
Delete	This button does not apply to this form type.
OK	This button does not apply to this form type.
Cancel	This button does not apply to this form type.
Add	This button does not apply to this form type.

Message Forms



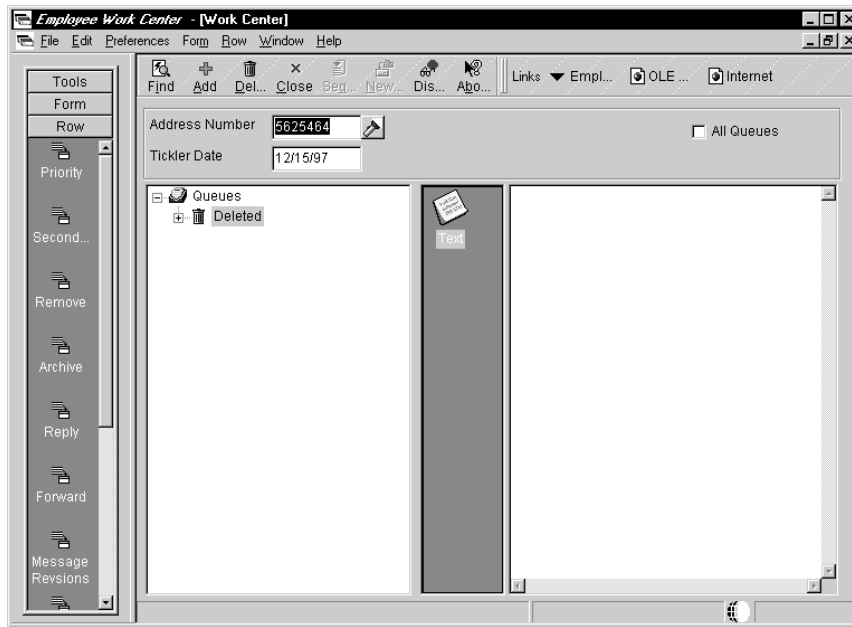
Use the Message form to display messages or request action from the user. The form is modal and is also not sizable. You can only add static text and pushbuttons to this form. This form is the only one that allows standard pushbuttons, including OK, Cancel, Yes, and No buttons. Do not use form interconnections on this form.

A delete confirmation is a good example of how you can use the Message form.

There is no business view for this form type.

There is no toolbar functionality for this form.

Parent/Child Forms



You can use the parent/child form to represent parent/child relationships in an application. The form has a parent/child control placed where the grid resides in a Find/Browse form. This control presents a tree view in the left portion of the control, which displays a visual representation of the parent/child relationship. The right portion of the composite control displays a grid in browse mode. A movable bar separates the tree view and grid. Use it to resize either view horizontally. The grid displays the detail information for the nodes of the tree. The parent/child form uses one business view.

There are two modes available on the parent/child form. You can choose the default mode or show all details. The default mode displays the detail for all nodes at the same level in the tree. As you switch levels in the tree, the grid lines change to reflect the level you are in. If you choose to show all details, there is a one-to-one correspondence between the nodes in the tree and the grid lines. You can also load pre-expanded trees so that if you click Find the tree will display all levels instead of just loading one level of nodes at a time. If you choose this option, it shows up as an option on the right mouse menu. You can also allow multiple selects.

Use the Parent/Child form to:

- Represent Parent/Child relationships.

- Perform normal Find/Browse functions (including FETCH and SELECT WHERE statements) and present information in a tree format.

The following table describes toolbar functionality for this form.

Select	Standard toolbar button that comes with the form. You must add the appropriate form interconnections to provide functionality.
Find	Standard toolbar button that comes with the form
Close	Standard toolbar button that comes with the form
Add	You can add this button to the toolbar. You must also provide the functionality for the button, which typically includes a form interconnect.
Copy	You can add this button to the toolbar. You must also provide the functionality for the button.
Delete	You can add this button to the toolbar. You must also provide the functionality for the button.
OK	This button does not apply to this form type.
Cancel	This button does not apply to this form type.

Parent/Child form processing is based on the concept of a parent/child relationship between two data items either within a table or across different tables. If the parent/child relationship is within a table or business view it is an inherent relationship. If the parent/child relationship is between two different tables or business views, then the relationship is an established one.

To display explicit parent/child data, the business view underlying the parent/child form should have a parent column and a child column. When any node on the tree for the form is expanded, a fetch is performed by querying the database for all the expanded node's child records. For example:

- Business unit 10 consists of business units 5 and 3.
- Business unit 5 consists of business units 2 and 1.

There is an inherent parent/child relationship between business unit 10 and business units 2 and 1.

If you simply wish to display data in a hierarchical fashion, the business view does not need to have a parent column and a child column. You must develop the logic to perform a fetch when a tree node is expanded.

The Delete hyperitem will delete the currently selected node record from the table. You must delete child records if needed.

The steps involved in creating a Parent/Child form are different than other form types. To create a Parent/Child form:

- Create the form in Form Design Aid
- Select a Business View
- Highlight the grid and add columns to the grid
- Add filter fields to the form

Choose the Parent field from the business view as the filter because the tool uses this for fetches from the database

Until this point, the form setup for Parent/Child is almost identical to the setup for a Find./Browse form. From this point on, there are unique steps to complete the Parent/Child form. Refer to *Working with Controls* for information about completing the Parent/Child form design.

Creating a Form

You use Forms Design to create one or more forms that display in an application. These forms are the visual interface for the end-user of your application and enable that user to view, add, or modify data stored in one or more tables.

To create a form, you must:

- Select a form type
- Define form properties
- Select a business view

After you create a form, you can also:

- Revise information about a form

Before You Begin

Before creating a form you must:

- ☐ Create a table if there is not an existing one you can use
- ☐ Create a business view upon which to base a form
- ☐ Add an application in the Object Librarian
- ☐ Understand the different form types
- ☐ Think about which data items are required for each form in the application



To select a form type

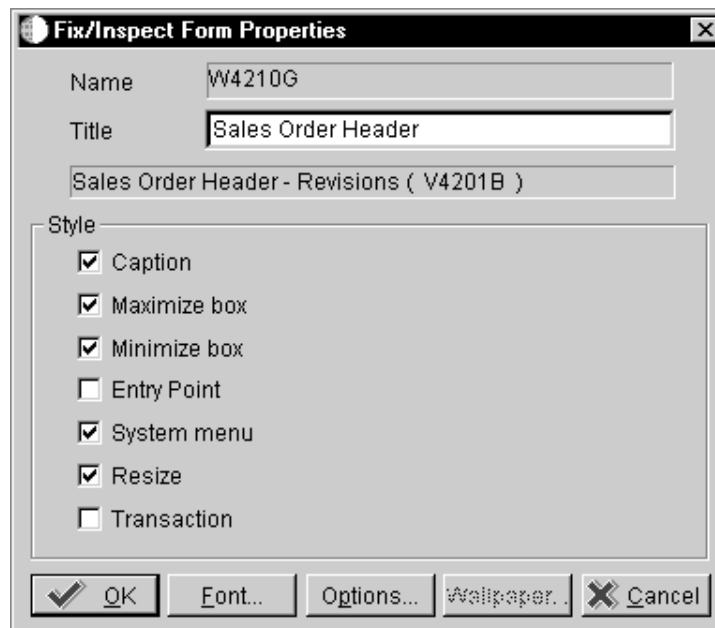
On Application Design

1. From the Form menu, choose Design.
2. From the Layout menu, choose Size to Guide to make your forms a standard size.

The form guide is a blue box that will appear.

3. From the Form menu, choose Create, and select one of the following form types:
 - Find/Browse
 - Fix/Inspect
 - Header Detail
 - Headerless Detail
 - Search and Select
 - Parent/Child
 - Message Form

The Window Properties form appears.



The business view name only appears in the gray area below the Title on Form Properties after you define your form properties and select a business view for your form.

To define form properties

On Window Properties

1. Complete the following field:
 - Title
2. Click one or more of the following Style options:
 - Caption

- Maximize/Minimize
- Entry Point
- System Menu
- Resize
- Transaction Processing

See *Transaction Processing* for instructions about defining boundaries for transaction processing.

Entry Point indicates that the form is the first form displayed when the application is accessed.

3. Click the following buttons to further alter or define the form:

- Font
- Options

Field	Explanation
Title	The title for the form, field, control, or button. • For a form: This will appear on the form at runtime. • For a field, control, or button: If the title is longer than the size of the control or button, it will be truncated. Resize the control to show the complete title.
Caption	Displays the form title at runtime.
Maximize/Minimize	Allows the user to minimize or maximize the form. When you turn on these options, the minimize and maximize buttons are displayed in the upper right corner of the form.
Entry Point	Designates the form as the initial form displayed in the application. There is only one entry point per application. If you are starting your application from a Fix/Inspect form, it does not need an entry point designated. Use this option to make the entry point form accessible from a menu.
System Menu	Includes the system menu button in the upper left corner of the form.
Resize	Allows the user to resize your form at runtime. Use this feature along with the modeless option.
Transaction	Enables transaction processing for this form. Transaction processing occurs within the specified transaction boundary so that database operations are stored in a queue until a commit command is issued.

J.D. Edwards Naming Standards

You should be aware of the following guidelines when you add a form.

Forms

Forms Design automatically assigns a name using the following format: **WzzzzzzzzA**

W = form

zzzzzzzz = is the application name

A = the first form created in the application. It is usually, but not always, the entry point to the application; subsequent forms are assigned sequential letters, such as B for the second form, C for the third and so on.

For example, the application P0101 has two forms. The first form, Work with Addresses, is the entry point and is assigned the name, W0101**A**. The second form, Address Book Revisions, is assigned the name W0101**B**.

Provide a form description based on the form type:

Find/Browse - *Work With* followed by the subject of the application, such as Work With Companies, Work With Constants.

Fix/Inspect, Header Detail and Headerless Detail - should reflect the topic they cover, such as Supplier Information, Item Master Revisions, Purchase Order Entry.

For lower level forms, identify the form that called the form by appending the calling form's title, such as Enter Voucher - G/L Distribution.

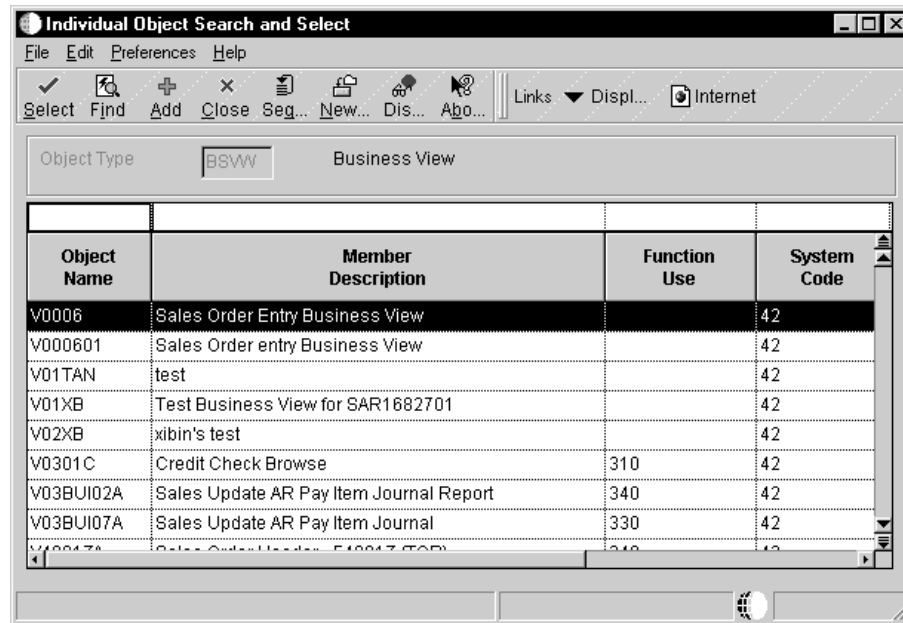
When the title of a form includes a verb, use an active verb instead of a nominalization, such as Work With Vouchers.



To select a business view

On the form with which you are working

1. From the Form menu, choose Business View.

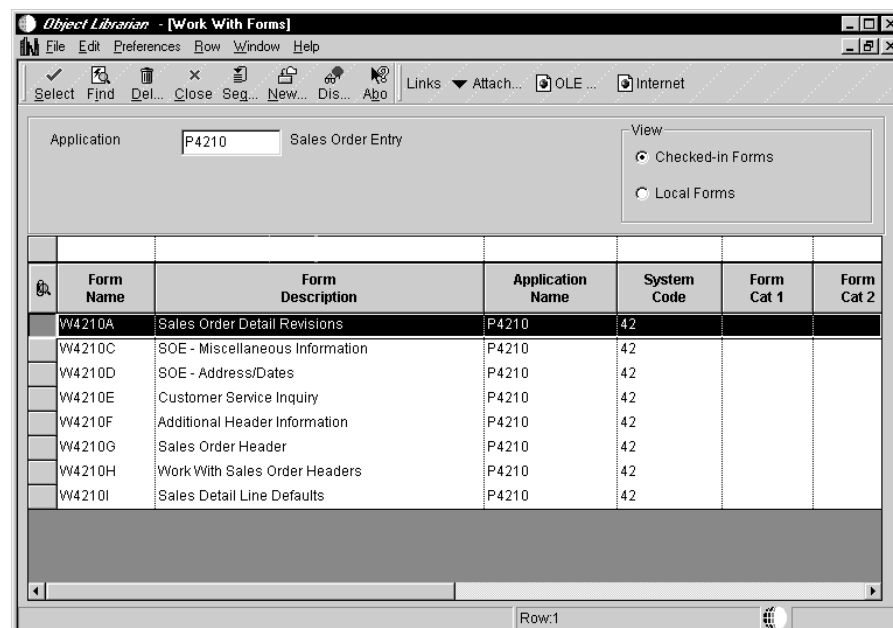


- Choose the business view to attach to the form.

► To work with forms

On Object Librarian

- Choose the application you wish to work with and click Select.
- On Application Design Aid, choose Named Forms from the Form menu.



3. On Work with Forms, choose one of the following options:

- Checked-in Forms
- Local Forms

When you create a form, it is local to your machine until you check the form in. When you check in the form, the F9865 file is updated. You can view forms that have been checked in or forms that are local to your machine.

4. Choose one of the available forms to view or revise more detailed information about the form.

The screenshot shows a window titled "Object Librarian - [Form Information Revisions]". It has a menu bar with "File", "Edit", "Preferences", "Window", and "Help". Below the menu bar is a toolbar with buttons for "OK", "Cancel", "Dismiss", "Apply", "Links", "Display", "OLE", and "Internet". The main area contains a list of fields for form information:

- Form Name: W4210A
- Form Description: Sales Order Detail Revisions
- Form Process Type: HD (with a dropdown arrow) - Header Detail Form
- Application: P4210
- System Code: 42 - Sales Management
- Help ID: 421001
- Help File Name: 42
- Category 1: (empty)
- Category 2: (empty)
- Category 3: (empty)
- Category 4: (empty)
- Category 5: (empty)

There is also a checkbox labeled "Entry Point" which is currently unchecked.

Field	Explanation
Form Process Type	The Form Process Type determines what the process type of the form is. For example: Find/Browse, Fix/Inspect and Header/Detail are all form process types. For a list of valid process types, see User Defined Code 98/FT.
Help ID	The Help ID identifies the index to the help file.

► To delete a form

On the form you want to delete

1. From the Form Menu, choose Destroy.
2. Confirm the deletion.



Once you delete a form, you cannot retrieve it.

Designing a Form Layout

Forms Design includes features that help you to define your form layout. This section highlights these features and provides you with information about:

- ☐ Selecting and moving controls
- ☐ Changing the size of grids, and controls
- ☐ Using Cut, Copy, and Paste in forms
- ☐ Aligning controls

Selecting and Moving Controls

Select and move controls as you design the form. You can:

- Move a control
- Select and move a group of controls
- Move a control and static text (and also any associated text)



To move a control

In Forms Design, on the form with which you are working

1. Position the mouse pointer over the control.

The pointer changes to the hollow square design tool.

2. Click and hold the mouse button as you drag the control to where you want it. Then release the mouse button.



To select and move a group of controls

On the form with which you are working

1. Place the mouse pointer outside the group of controls you want to select.
2. Press and hold the mouse button. The pointer changes to the pencil design tool.

3. Drag the mouse to the opposite corner of the group of controls.

As you drag the mouse, a rectangle is drawn from the point you first placed the arrow design tool. Make sure the rectangle completely surrounds all the controls you want to select.

4. Release the mouse button. The selected group of controls is surrounded by a box.
5. Position the mouse pointer over the selected group.

The pointer changes to the hollow square design tool.

6. Click and hold the mouse button as you drag the group to where you want it. Then release the mouse button.

To move a control and static text

On the form with which you are working

1. Position the mouse pointer over the shaded area between the static text and the control box.
2. When the pointer turns into the hollow square design tool, press and hold the mouse button. When the crosswire appears, you can move both the static text and the control box together.
3. Drag the controls to where you want them. Release the mouse button.

If the static text and the control box are too close together, the hollow square design tool will not appear in the shaded area. If this occurs, move the text and the control further apart. You can also use the pencil tool to draw a box around just one control and move it independently.

Changing the Size of Grids, and Controls

There are two methods you can use to change the size of a control:

- Size controls using the mouse
- Size and place a control using the Size command

The Size command allows you to specify placement on the form. You can also specify size and placement for only one control at a time.

J.D. Edwards Design Standards

- The length of string fields must be at least the minimum size of the string of characters.

► **To size controls using the mouse**

On the form with which you are working

1. Choose the control. Handles appear on the control.
2. Move the mouse pointer over one of the handles.

The mouse pointer changes to the filled-square design tool.

3. To change the width, drag a handle at the left or right of the control.

To change the height, drag a handle at the top or bottom of the control.

To change both the width and the height, drag a handle at the corner of the control.

Resize the form and grid using the same method.

► **To size and place a control using the Size command**

On the form with which you are working

1. Choose the control.
2. From the Layout menu, choose Size.

Resize Control

Height Top

Width Left

Width Calculator

☒ Average ☐ Maximum Characters

3. To specify the exact control size in pixels, complete the following fields:

- Height/Width
4. To have the tool automatically calculate the width of a control, type the number of characters in the Width Calculator field, and click one of the following options:
 - Average
 - Maximum Characters

Average width is about the size of a lowercase x. Maximum width is about the size of an uppercase W.

5. Click the Calculate button.

The new calculated width, in pixels, appears in the Width field.

6. Click OK to accept the width.

The Resize Control window remains open for additional changes.

7. To specify the exact position of the control on the form, complete the following fields in pixels:
 - Top
 - Left

Pixels are calculated from the top and left margins of the form.

8. Click Close when you are finished.

Field	Explanation
Height/Width	Specifies the control measurements in pixels.
Top/Left	Designates the placement, in pixels, of a control on the form.
Width Calculator	Calculates the size of a control. To calculate the size, indicate whether the value in the Width Calculator is an average or maximum number of characters. Click on Calculate and the width is automatically adjusted. The new width in pixels appears in the width field.
Average	Used to calculate and resize the width of a control. The new width is calculated using the value you enter in the Width Calculator field.
Maximum Characters	Resizes the width of a control based on the value you enter in the Width Calculator field.

Calculate

Calculates the size of a control. To calculate the size, indicate whether the value in the Width Calculator is an average or maximum number of characters. Click on Calculate and the width is automatically adjusted. The new width in pixels appears in the width field.

Using Cut, Copy, and Paste in Forms

You use the Cut, Copy, and Paste commands from the Edit menu to move and copy controls within the same form or on another form within the same application. You can also create a new form or application into which to paste the controls.

To cut, copy, and paste controls

On the form with which you are working

1. Choose the control or group of controls.
2. From the Edit menu, choose Copy or Cut.
3. Select the form into which you want to paste the control or group of controls.
4. From the Edit menu, choose Paste.
5. Move the outline of the controls to the desired location on the form.
6. Click once to place the controls.

Event rules attached to a control are copied, cut, or pasted when you copy, cut, or paste the control. You cannot copy or paste complex controls such as a tab control, grid control, or grid column.

Aligning Controls

Use the Layout menu to:

- Align a group of controls
- Use an alignment grid to align controls

To align a group of controls

On the form with which you are working

1. Select the group of controls you want to align.

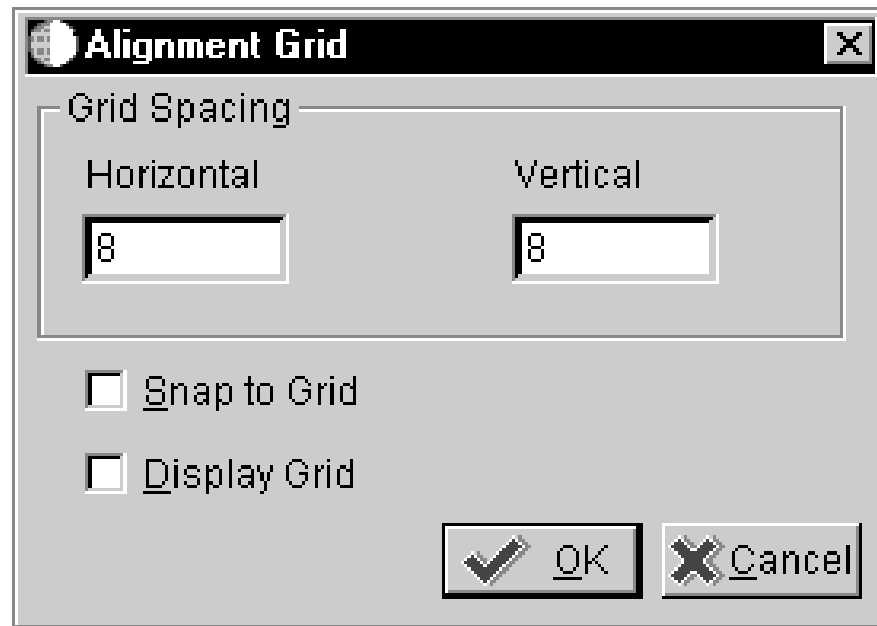
2. Click the control within the group to which you want to align the group of controls.
3. From the Layout menu, choose one of the following commands:
 - Align Left
 - Align Center
 - Align Right
 - Align Top
 - Align Middle
 - Align Bottom
 - Align Database

Field	Explanation
Align Left	Aligns a group of controls vertically with the left edge of the selected control.
Align Center	Centers a group of controls vertically in relation to the selected control.
Align Right	Aligns a group of controls vertically with the right edge of the selected control.
Align Top	Aligns a group of controls horizontally with the top edge of the selected control.
Align Middle	Centers a group of controls horizontally in relation to the selected control.
Align Bottom	Aligns a group of controls horizontally with the bottom edge of the selected control.
Align Database	Aligns a group of database or dictionary items. This left aligns the static text controls with the selected control and automatically aligns the associated control boxes as well.

 To use an alignment grid to align controls

On the form with which you are working

1. From the Layout menu, choose Grid Alignment.



2. To determine grid spacing, complete the following fields:
 - Vertical/Horizontal Placement
3. Click one or both of the following:
 - Snap to Grid
 - Display Alignment Grid

The following example shows a form displaying the alignment grid.

Field	Explanation
Vertical/Horizontal Placement	Specifies the space between the horizontal and vertical grid coordinates. The default spacing is 8 pixels between each horizontal and vertical gridline.
Snap to Grid	Places the control to the closest vertical or horizontal grid line when you move the control on the form.
Display Alignment Grid	Displays the alignment grid on the form in design mode.

Using Undo and Redo

You use undo to undo the last change you made. You can set the depth on the undo feature to undo many (over 100) changes back. However, a large setting requires a large buffer size and may lead to performance degradation.

► To set the undo depth

On Form Design

1. From the edit menu, choose Set Undo Depth.
2. Enter a number to indicate how many steps back you can undo.

► To undo and redo changes

On Form Design

1. To undo a change (that is to move backward in the stack of changes), from the Edit menu, choose Undo.

Each time you choose Undo, the tool reverts back by one change.

2. To reapply a change (that is, to move forward in the stack of changes), from the Edit menu, choose Redo.

Each time you choose Redo, the tool reapplies one change.

Creating Menu/Toolbar Exits

Menu/Toolbar Exits can be items that are only in the menu or items that appear in the menu and on the toolbar. An item must be placed in the menu in order to appear on the toolbar. The toolbar is a visual shortcut to menu items. You can create custom exits, or you can use any of the standard exits OneWorld provides for a form. You can also modify these standard exits.

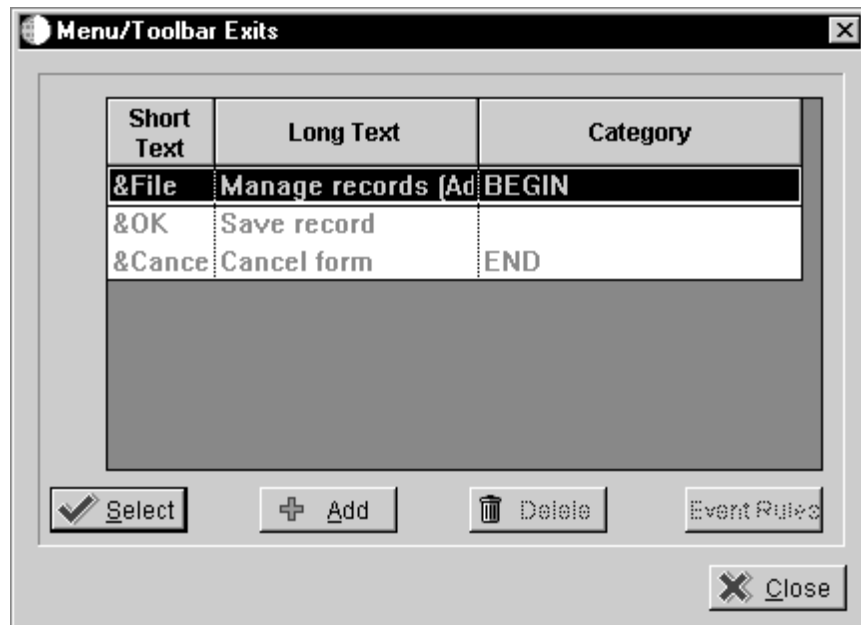
To create a Menu/Toolbar exit, you:

- Designate the exit category
- Define the exit properties
- Attach event rule logic if needed
- Select a bitmap for the exit if you are going to show the item on the toolbar

► To create, edit, or delete a Menu/Toolbar exit

In Form Design, on the form with which you are working

1. From the Form menu, choose Menu/Toolbar Exits.



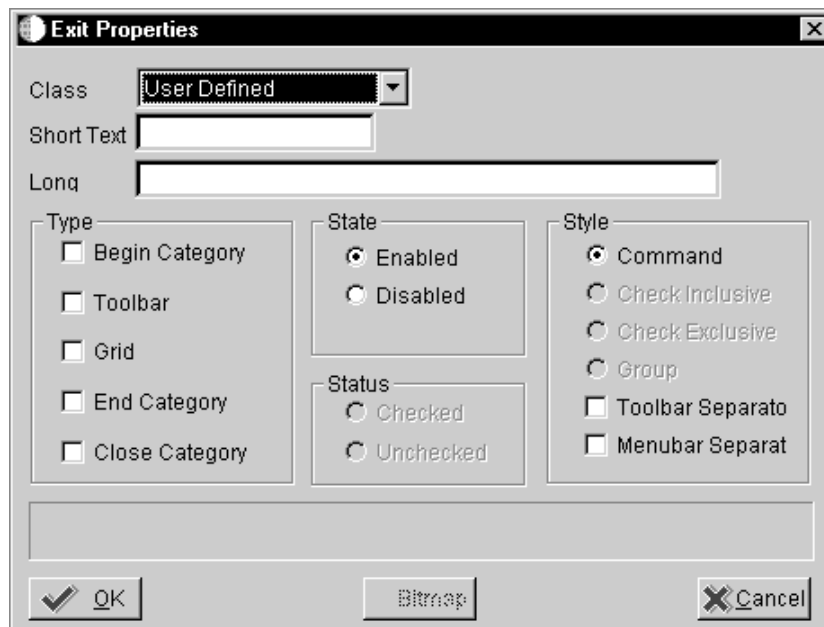
Depending on the form type you are working with, several standard exits will appear. These might include:

OK	Accepts the data in the form, clears fields, and remains in the form. However, in update mode OK closes the form. In add mode OK may or may not close the form.
Cancel	Closes the form and returns to the previous one. Any additions, revisions, or deletions the user made in this form are ignored.

2. If you do not want one of the listed exits on your form, choose the exit and click Delete.
3. Choose an exit and click select to edit its properties.
4. To create a new exit, click Add.
5. On Exit Properties, choose the appropriate class.

Classes available depend on the form type. The classes and choices available on the Exit Properties form will also vary depending on the type of exit you select.

Form, row, and view are standard exits and are the only Begin Categories you use. All exits under these categories are user defined, and the last one in the list requires an End Category. If you select a Row Category, all exits under it need the Grid option turned on.



The image shows a dialog box titled "Exit Properties". It contains several fields and groups of options:

- Class:** A dropdown menu currently showing "User Defined".
- Short Text:** An empty text input field.
- Long:** An empty text input field.
- Type:** A group box containing five checkboxes:
 - ☐ Begin Category
 - ☐ Toolbar
 - ☐ Grid
 - ☐ End Category
 - ☐ Close Category
- State:** A group box containing two radio buttons:
 - ☒ Enabled
 - ☐ Disabled
- Status:** A group box containing two radio buttons:
 - ☐ Checked
 - ☐ Unchecked
- Style:** A group box containing several options:
 - ☒ Command
 - ☐ Check Inclusive
 - ☐ Check Exclusive
 - ☐ Group
 - ☐ Toolbar Separato
 - ☐ Menubar Separat

At the bottom of the dialog are three buttons: "OK" (with a checkmark icon), "Bitmap" (disabled), and "Cancel" (with an 'X' icon).

6. Complete the following fields:

- Short Text

The short text includes the access key.

- Long Text

The long text appears in the status bar.

7. Complete the Exit Properties form and click OK.

Every Begin category must have a corresponding End category. There can be many exits between the Beginning and End categories.

Field	Explanation
Short Text	The short description of the item that appears on the menu and exit bar.
Long Text	The long description of the hyper-item that appears on the menu.
Begin Category	Indicates a category which will appear on the menu bar, such as Row, Form, Report, and View or as a title of cascading (sub) menu.
Toolbar	Specifies that the hyper-item should be placed on the toolbar.
End Category	Specifies that the hyper-item is the last item in this category.
Close Category	Nests hyper-exits for multiple hyper-categories. Closes all open Begin Categories above it.
Enabled	Specifies that the item is initially enabled. (You can then use a system function to disable the item.)
Disabled	Specifies that the item is initially disabled.
Checked	Specifies that the item is initially checked. This option works only if the hyper-item has been defined with a Check Inclusive/Exclusive flag set. (future)
Unchecked	Specifies that the item is initially unchecked. This option works only if the hyper-item has been defined with a Check Inclusive/Exclusive flag set. (future)
Command	Defines the item as a command button. Leave it unselected to define the hyper-item as a menu item.
Check Inclusive	Specifies whether the item is part of a group. Inclusive means that multiple checks are allowed within the group. (future)
Check Exclusive	Specifies whether the hyper-item is part of a group. Exclusive means that multiple checks are not allowed within the group. (future)

Field	Explanation
Group	When enabled, this specifies that the hyper-item is a group item.
Toolbar Separator	Specifies that a separator should be placed on the toolbar before this hyper-item.
Menubar Separator	Specifies that a separator should be placed on the menubar after this hyper-item.

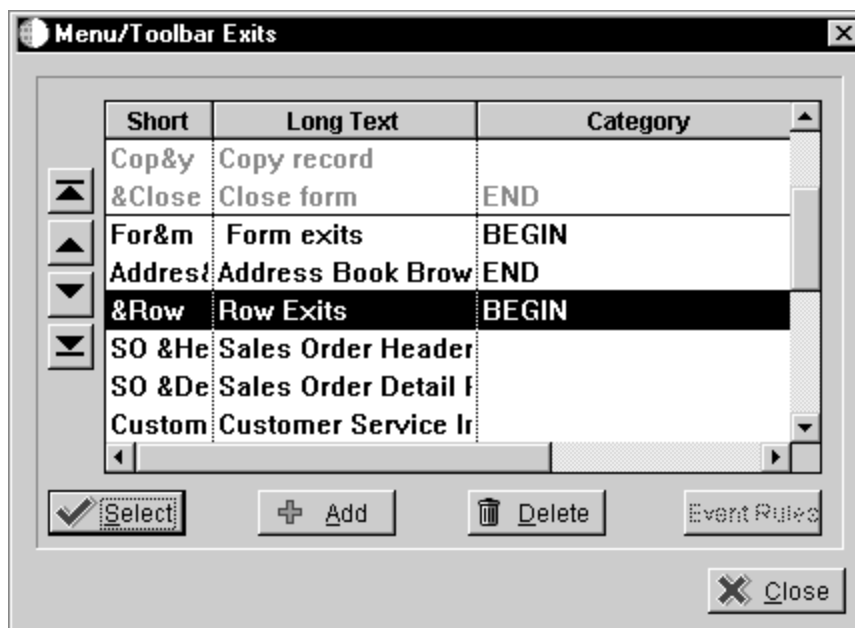
Adding Bitmaps

No bitmap strips initially appear for you to choose. Bitmap strips are stored in the `\package name\res` directory, for example, the `b7\appl_pgf\res` directory. Each bitmap strip is named starting with `hc_`. After you have set that strip on an exit for a Begin category it appears for any item listed under that category. You can also use third party tools to modify and initially create your bitmaps.

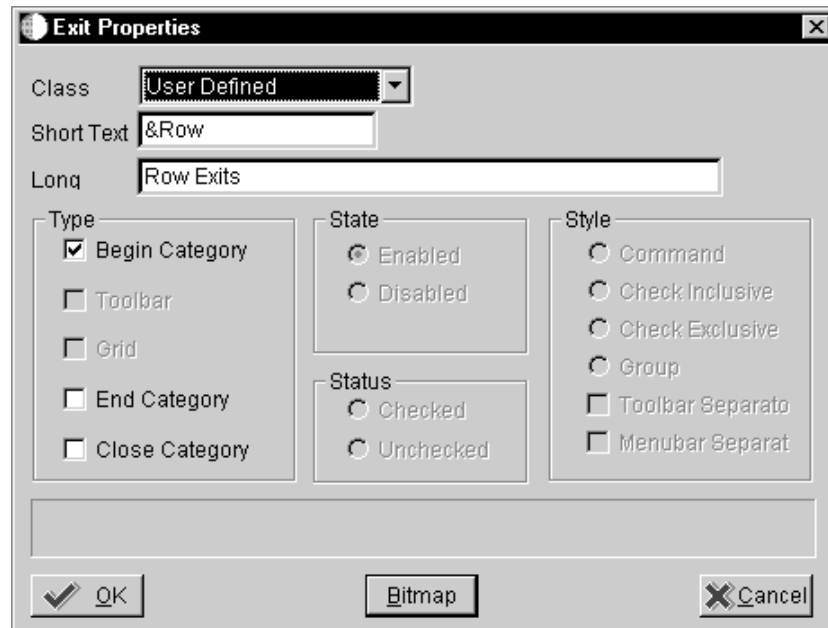
► To add a bitmap

On Menu/Toolbar Exits

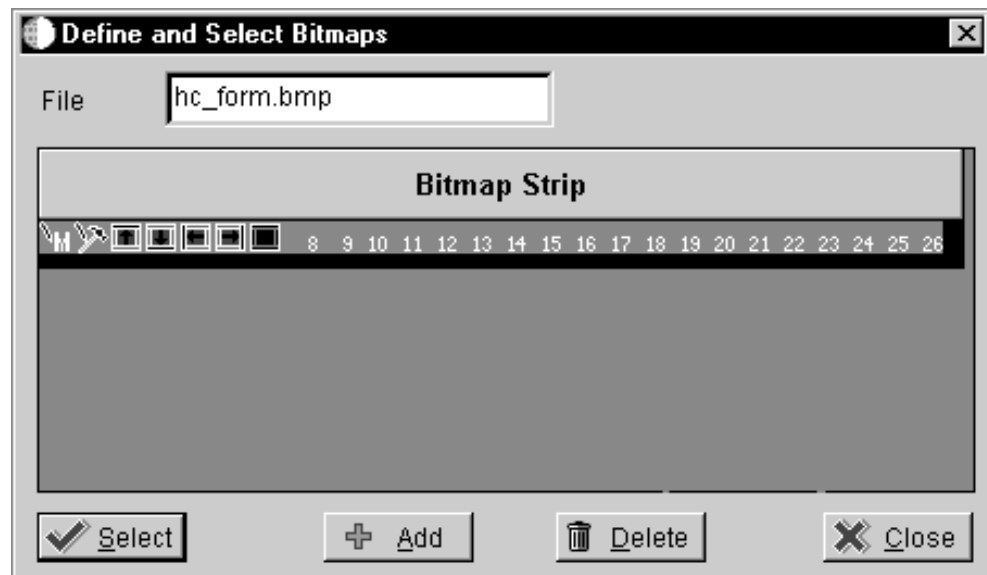
1. Choose a Begin Category.



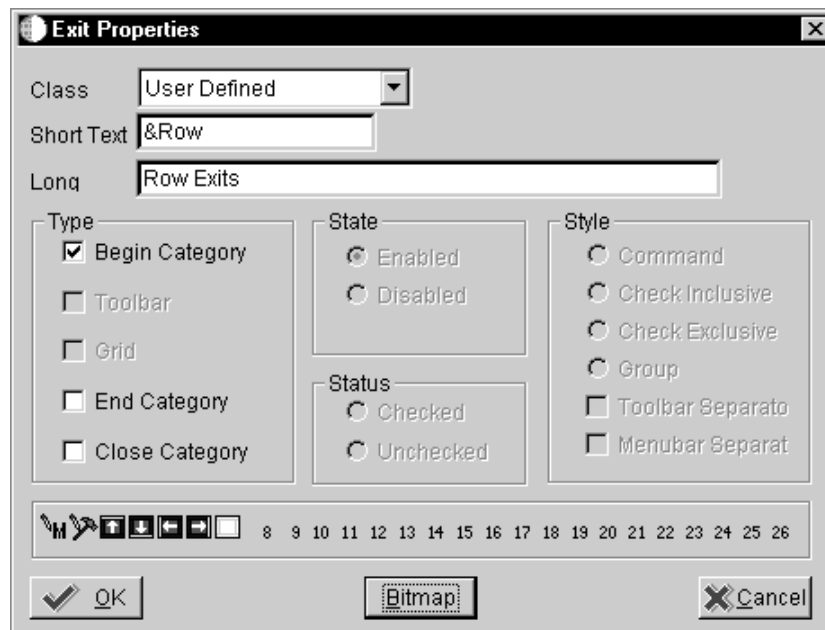
The Exit Properties form appears.



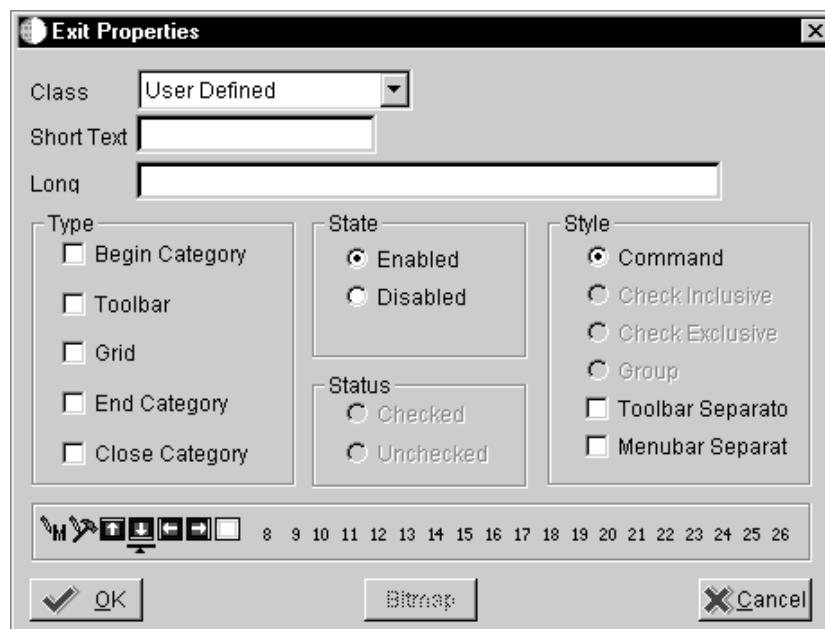
2. On Exit Properties, click the Bitmap button.



3. Type the .bmp file name for your bitmap strip in the File field.
4. Click Add.
5. Choose the strip you wish to use and click Select to apply to all the exits under that category.



6. On the Exit Properties form click on the specific bitmap that you want to use for each exit.



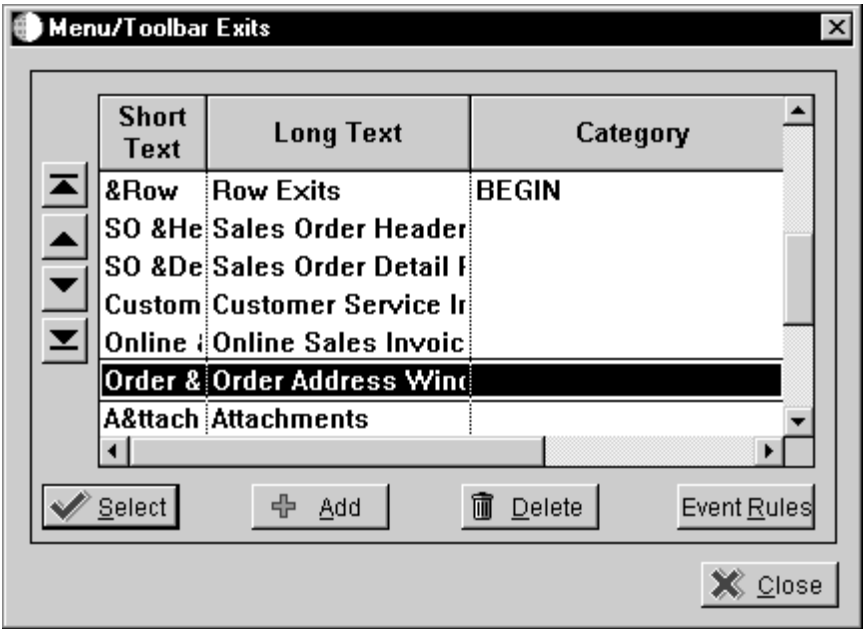
Attaching Event Rules to a Menu/Toolbar Exit

You cannot attach event rules to Begin Categories. See *Event Rules Design* for more information about creating event rules.

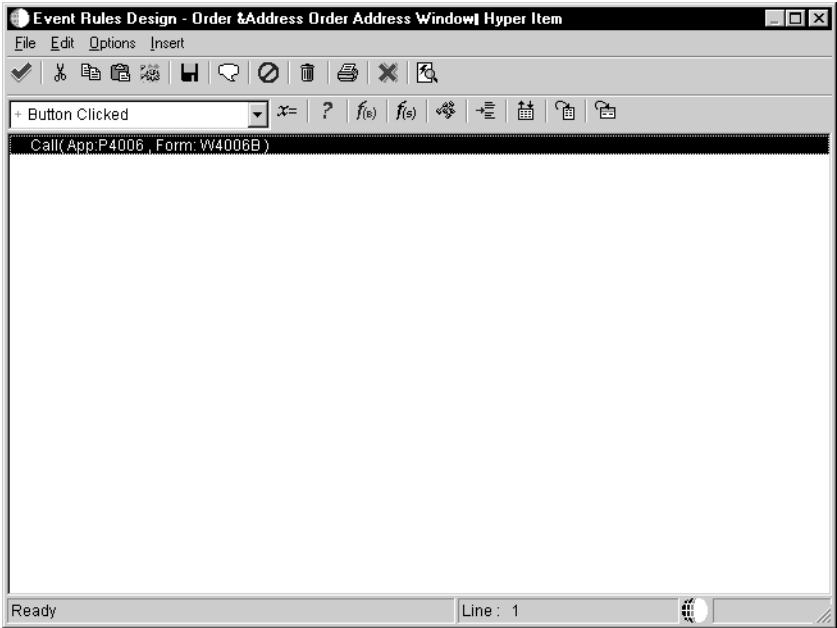
► To attach event rules to a Menu/Toolbar Exit

On the Menu/Toolbar Exits form

- 1. Select the exit you wish to use and click the Event Rules button.



- 2. On Event Rules Design choose the Button Clicked event.
- 3. Add event rule logic you wish to attach.



4. Save, and exit Event Rules Design.

Creating Subcategories in Menus

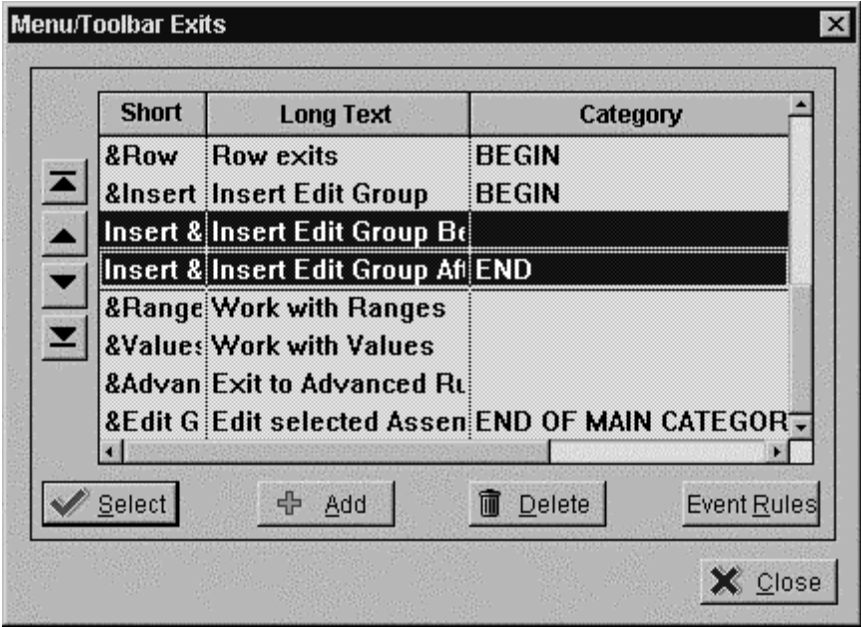
OneWorld allows you to create subcategories for drop down items (cascading menus):

The screenshot shows the 'Assembly Inclusion Rules' window with the 'Edit' menu open. The 'Edit' menu has options: 'Insert Edit Group', 'Insert Before', 'Insert After', 'Ranges', 'Values', 'Advanced Rules', and 'Edit Group'. The 'Edit Group' option is selected, and a sub-menu is visible with 'Insert Edit Group' and 'Insert After' options. The 'Configured Item' field is set to '6000'. The 'Rule Type' is set to 'P'. The 'Parts List' is visible. Below the menu, there is a table with columns: 'Op', 'Segment Number', 'Rel', 'Values', 'Item', and 'Item Description'.

Op	Segment Number	Rel	Values	Item	Item Description
*				6100	Forklift, Boom
*				6013	Counter Weights
I	50	EQ	25		
*				6011	25lb Propane Tank
I	50	EQ	50		
*				6012	50lb Propane Tank
I	10	EQ	6000		
A	30	GE	10		
*				6014	Pnematic Tire
E				6015	Hard Rubber Tire
I	30	EQ	08		
*				6007	8ft Boom
I	30	EQ	10		

► To create a subcategory in a Menu

1. Choose the Menu/ToolBar exits within Form Design Aid and “nest” the Begin and End for that form exit.



Insert Edit Group Before and Insert Edit Group After are further subcategories of the Insert Edit Group menu item.

- 2. Define event rules for each of the exits.

Working with Controls

Use form controls to provide specific functions within an application. For example:

- Insert field controls to display data, enter data, calculate data, store data permanently or temporarily, or pass data between other fields and forms.
- Place check boxes to provide for multiple selections, or radio buttons to indicate mutually exclusive selections.



You can have a maximum of 250 controls on a form. You can use Form Properties to check the number of controls on a form. You will also receive a warning if you are near the 250 control limit.

Working with controls includes the following tasks:

- ☐ Understanding form controls
- ☐ Creating push buttons
- ☐ Creating check boxes
- ☐ Creating radio buttons
- ☐ Adding data items to a form
- ☐ Creating static text controls
- ☐ Creating edit controls
- ☐ Creating UDC edit controls
- ☐ Creating media objects
- ☐ Creating bitmap controls
- ☐ Creating tree controls
- ☐ Creating group boxes
- ☐ Defining grid properties
- ☐ Defining grid column properties
- ☐ Defining properties for parent/child controls

- ☐ Defining options
- ☐ Associating database items, dictionary items, or descriptions
- ☐ Changing tab sequence
- ☐ Creating tab controls
- ☐ Overriding data dictionary triggers at design time
- ☐ Designing forms using multiple modes

Before You Begin

Before you work with form controls, you must:

- ☐ Create a form

Understanding Form Controls

Each form automatically includes certain default controls, depending on the type of form you are creating. However, you might need to add additional controls as you design the form. Choose from standard Windows graphical controls as well as J.D. Edwards custom controls. Available controls include the following:

- Push Button
- Check Box
- Radio Button
- Edit
- Static Text
- UDC Edit
- Grid
- Parent/Child
- Media Object
- Group Box
- Bitmap
- Tree Control
- Tab Control
- Business View Field
- Data Dictionary Field

You can disable controls at design time and then enable them at runtime using event rules. If a control is disabled, it is grayed out. If a control is read-only, you can set focus on it or tab into it, but you cannot enter values.

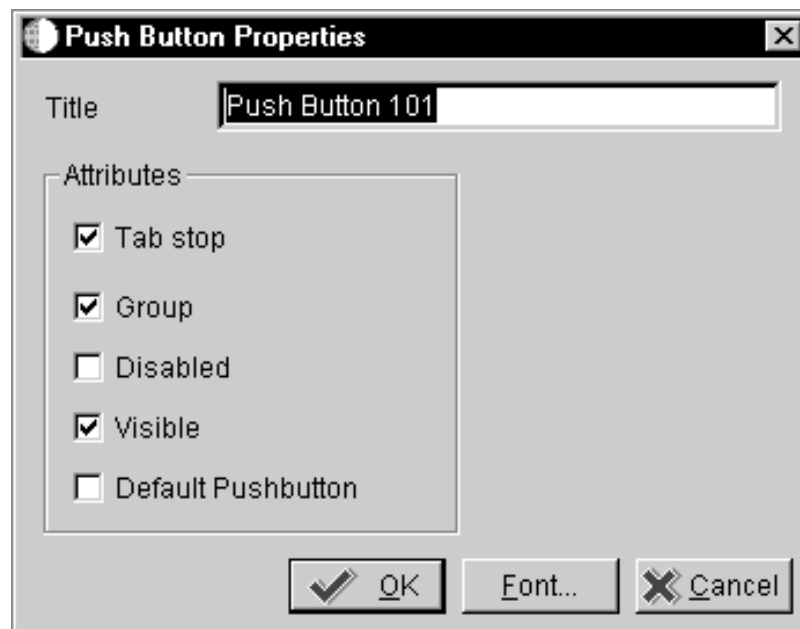
Creating Push Buttons

You can use a push button to initiate an action or a set of actions.

► To create a push button

On the form with which you are working

1. From the Insert menu, choose Push Button.
2. Position the control outline on the form and click.
3. To define the push button properties do one of the following:
 - Double-click the push button control
 - Choose the push button and choose Item Properties from the Edit menu.



4. Complete the following field:
 - Title
5. Click one or more of the following Attributes:
 - Tab Stop

- Group
- Disabled
- Visible
- Default Push Button

6. Assign an access key by inserting the ampersand symbol (&) in front of the particular letter in the push button name (optional).

An access key provides a quick way to execute a push button using a combination of the Alt key and the specific alpha key. An access key is identified by any underlined letter within the push button name. Pressing the Alt key and that letter on your keyboard simultaneously is the same as clicking the button with your mouse.

Field	Explanation
Title	The title for the form or control. If the title is longer than the size of the control it will be truncated. Resize the control to show the complete title. For event rules title, this is the default name of the control for use in an event rule. The event rules title defaults to the static text associated with an item. If there is no associated static text and you want to use this control in an event rule, enter a name.
Tab Stop	Allows the user to tab into this control.
Group	Defines this control as the first in a series of related controls. A group is defined as this control plus all subsequent controls without the Group option turned on. The next control with Group turned on signals the start of a new group (a single control can form a group). Within a group of more than one control, use the arrow keys to move from one control to another instead of pressing Tab.
Disabled	Disables a control. A disabled control is grayed out at runtime.
Visible	Makes a control visible.
Default Push Button	Executes the selected push button upon pressing the Enter key.

Creating Check Boxes

You use check boxes to indicate choices. A check mark in a box indicates that you have made a choice. Check boxes are not mutually exclusive.

- You must associate a check box with a database or dictionary item.

- You can use a check box to pass a value to an event rule.

See Also

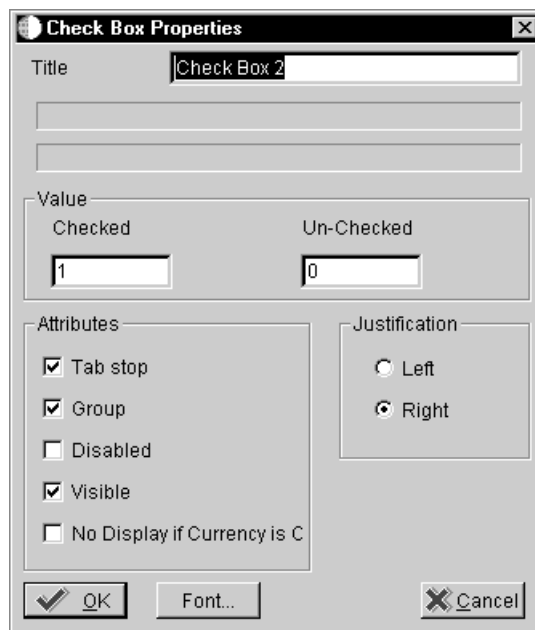
- *Associating Database Items, Dictionary Items, or Descriptions*



To create a check box

On the form with which you are working

1. From the Insert menu, choose Check Box.
2. Position the control outline on the form and click.
3. To define the check box properties do one of the following:
 - Double-click the check box control
 - Choose the check box control and choose Properties from the Settings menu.



4. Complete the following field:
 - Title
5. Complete the following Value fields:
 - Value Checked/Unchecked
6. Click one or more of the following attributes:
 - Tab Stop

- Group
- Disabled
- Visible
- No display if currency is OFF

7. Click one of the following Justification options:

- Left
- Right

Field	Explanation
Value Checked/Unchecked	Specifies the value used if this check box is selected. If a check box is associated with a database item, then the table is updated with this value. Otherwise, the value can only be used in an event rule.
Justification	Determines how the values entered at runtime will be justified within the edit box.
No display if currency is OFF	Hides this option if the Multicurrency flag = No in System Setup, General Accounting Constants (P0000).

Creating Radio Buttons

You use radio buttons to indicate choices. A filled radio button indicates that you have made a choice. If radio buttons are in a group box, they should always be mutually exclusive.

- You can associate radio buttons with a user defined code field, where each button has a value from the User Defined Code table.
- You associate a radio button with a database or dictionary item.

See Also

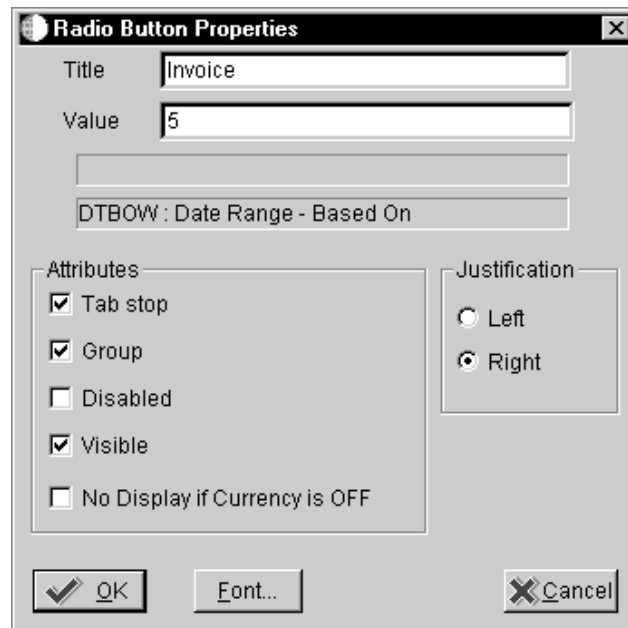
- *Associating Database Items, Dictionary Items, or Descriptions*

To create a radio button

On the form with which you are working

1. From the Insert menu, choose Radio Button.
2. Position the control outline on the form and click.

3. Repeat steps 1 and 2 for as many radio buttons as you need, then select all of the radio buttons to group them and associate each of them to make them mutually exclusive.
4. To define the radio button properties do one of the following:
 - Double-click on the radio button control
 - Choose the radio button control and choose Properties from the Settings menu.



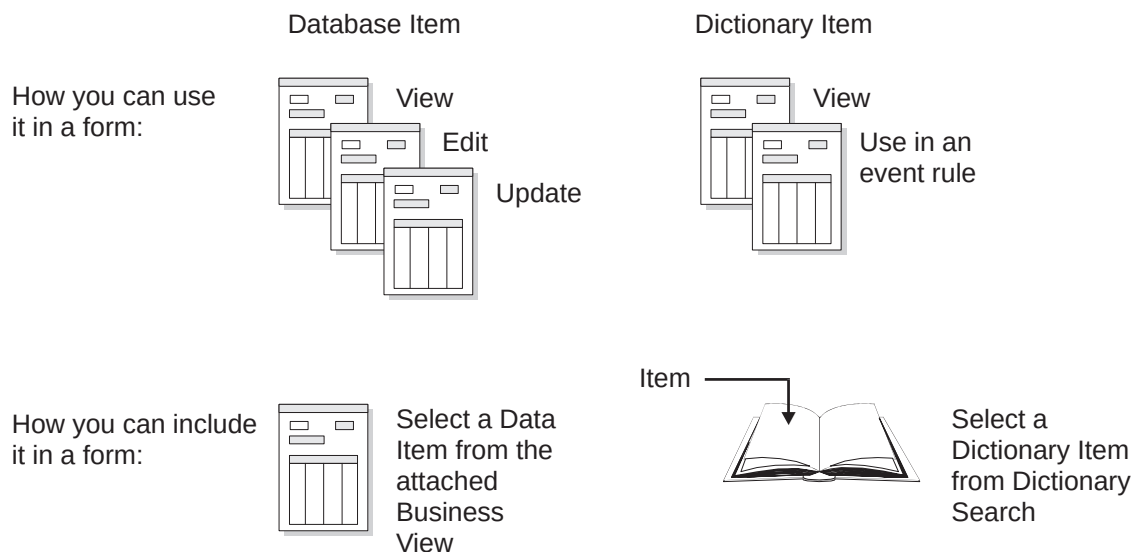
5. Complete the following fields:
 - Title
 - Value
6. Click one or more of the following Attributes:
 - Tab Stop
 - Group
 - Disabled
 - Visible
 - No display if currency is OFF
7. Click one of the following Justification options:
 - Left
 - Right

Field	Explanation
Value	Specifies a literal value that is used when the operator selects this radio button.

Adding Data Items to a Form

On a form, you can use any data item that is in the business view and any data item from the data dictionary. Database items are in the business view associated with the form. Data dictionary items are not in the business view. Use a data dictionary item to store information not contained in a database field. Only database items are updated to the database. On a form, database items appear with a blue box in the left corner and dictionary items appear with a yellow box in the left corner.

The following illustration compares the use of database items and data dictionary items in an application.



Inserting Database Items Onto a Form

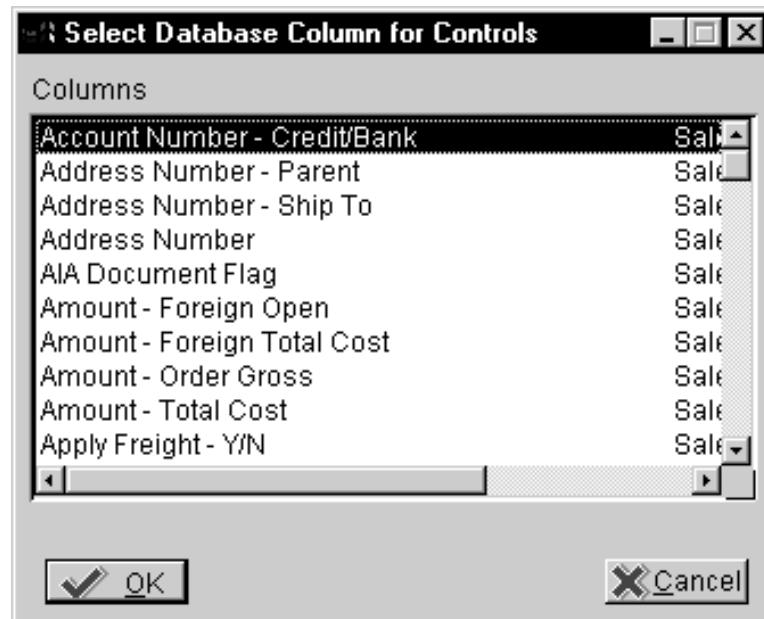
You can insert any data item from the business view to include on the form as a database item. When you insert a database item, it always includes both:

- A static text control that displays the row or column description from the data dictionary
- The associated edit control or UDC edit control

► To insert database items onto a form

On the form with which you are working

1. From the Insert menu, choose Business View Field.



2. Choose the database item.
3. Position the control outline on the form and click.

If the form has a grid, you can place a database item on a blank area of the form or in the grid. If you want to put the item in the grid, focus on the grid before you choose the database menu option.

The database control is marked with blue to distinguish it from a dictionary item, which is marked with yellow.

4. Continue to choose database items from the list. Click Cancel when you are finished.

See Also

- *Using Quick Form* in this guide.

Inserting Data Dictionary Items Onto a Form

You insert a dictionary item when it is not already included in the business view for the form or when you do not want to update the field in a table. Data dictionary items appear with a yellow box in the left corner on a form.

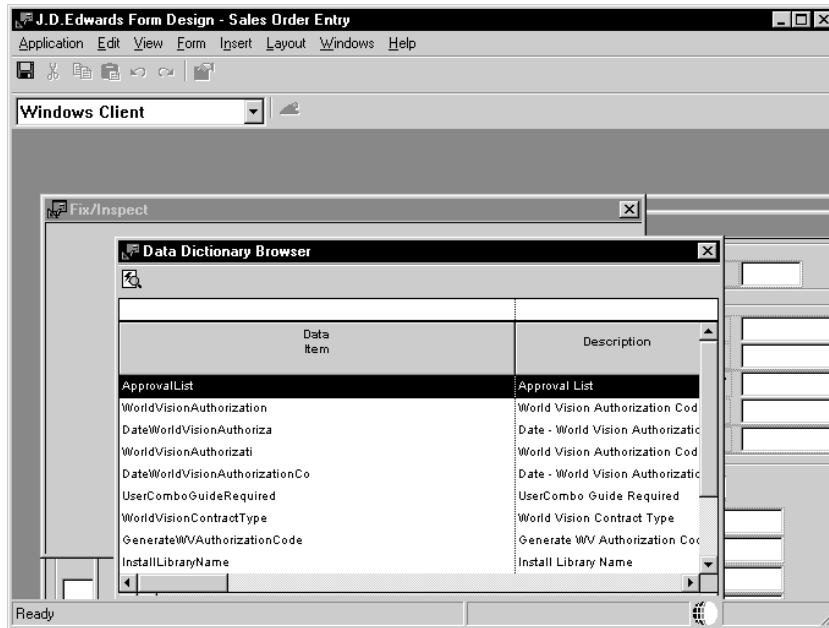
Use data dictionary items as display-only fields in a form or as a reference for an event rule. Data dictionary items do not retrieve or update data in a table.

► To insert data dictionary items onto a form

On the form with which you are working

1. From the Insert menu, choose Data Dictionary Field.

The Data Dictionary Browse Control form appears.



2. On Data Dictionary Browse Control, specify search criteria and press enter.
3. Choose a data dictionary item from the available items and drag it to your form.
4. Position the control outline on the form and click.

If the form has a grid, you can place a data dictionary item either on a blank area of the form or in the grid. If you put the item in the grid, focus on the grid before you choose the dictionary menu option.

The data dictionary control has a yellow box in the left corner to distinguish it from a database item, which has a blue box in the left corner.

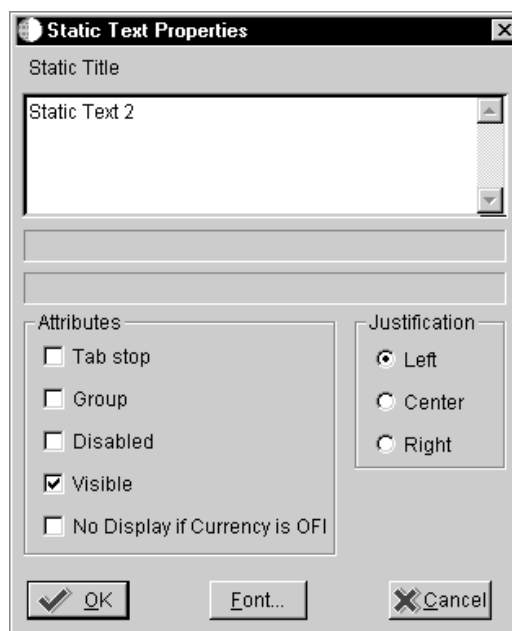
Creating Static Text Controls

Use a static text control to display descriptive text such as a titles or instructions. This text is not associated with a control, and the user cannot change it. It can be changed using the *Set Control Text* system function in event rules.

► **To create a static text control**

On the form with which you are working

1. From the Insert menu, choose Static Text.
2. Position the control outline on the form and click.
3. To define the static text control properties do one of the following:
 - Double-click the static text control.
 - Choose the static text control and choose Properties from the Settings menu.



4. Complete the following field by typing in the text you want to appear on the form:
 - Title
5. Click one or more of the following Attributes:
 - Tab Stop
 - Group
 - Disabled
 - Visible
 - No display if currency is OFF
6. Click one of the following Justification options:
 - Left

- Center
- Right

Creating Edit Controls

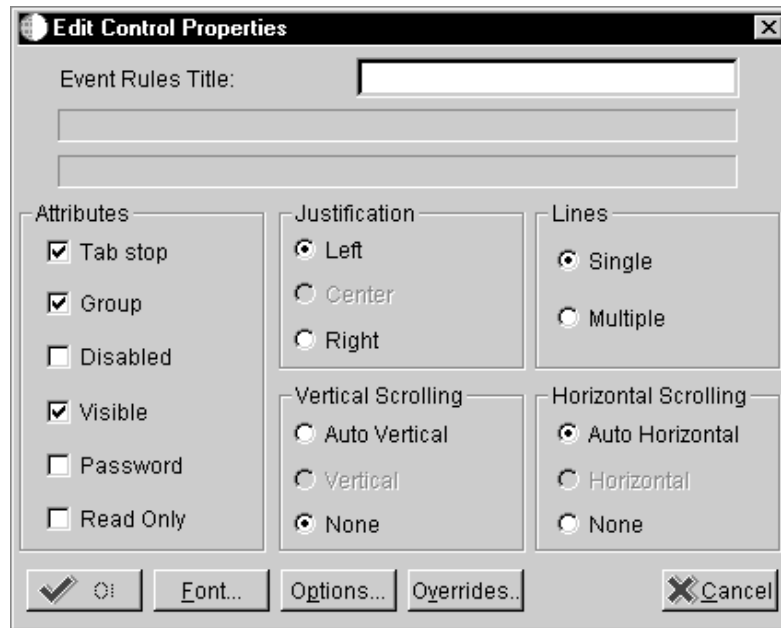
Edit controls are generic input fields and have no associated text. You should associate edit controls with database or dictionary items.

- If you associate an edit control with a database item, then the value entered by a user at runtime updates the table.
- If you associate an edit control with a dictionary item, then the value entered at runtime is for display only.
- Edit Controls have type ahead functionality. Once a user enters a character in the field, a history list is searched for a match. If there is a match, it is displayed in the field with highlighted text. This is particularly useful for data entry work because it can reduce the amount of typing required.

To create an edit control

On the form with which you are working

1. From the Insert menu, choose Edit.
2. Position the control outline on the form and click.
3. To define the edit control properties do one of the following:
 - Double-click on the edit control
 - Choose the edit control and choose Properties from the Settings menu.



4. Complete the following field:
 - Event Rules
5. Click one or more of the following Attributes:
 - Tab Stop
 - Group
 - Disabled
 - Visible
 - Password
 - Read Only
6. Click one of the following Justification options:
 - Left
 - Center
 - Right
7. Click one of the following Lines options:
 - Lines Single
 - Lines Multiple
8. Click one of the following Vertical Scrolling options:
 - Auto Vertical
 - Vertical
 - None
9. Click one of the following Horizontal Scrolling options:

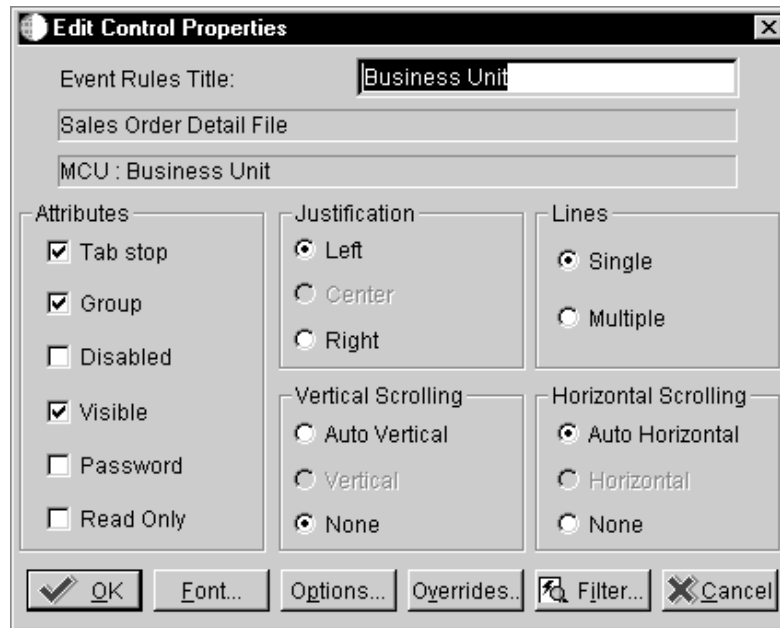
- Auto Horizontal
- Horizontal
- None

10. Click the following buttons to further alter the control:

- Font
- Options
- Overrides

Field	Explanation
Event Rules Title	The default name of the control for use in an event rule. It defaults to the static text associated with an item. If there is no associated static text and you are planning to use this control in an event rule, enter a name.
Password	Allows you to enter an asterisk (*) so you don't see the actual password.
Read Only	Prevents a user from changing the value in a field.
Justification	Determines how the values entered at runtime will be justified within the edit box.
Lines Single	Designates the edit control to be for a single line only.
Lines Multiple	You can make an edit control multi-line only if both automatic scroll options are off.
Scroll Bar Horizontal/Vertical	Displays the horizontal and/or vertical scroll bar. The Auto Vertical option scrolls the text vertically. The text scrolls up a full page when the user presses Enter on the last line of the edit text control. For this option to have any effect, you must also turn on the Multiple Lines radio button. The Auto Horizontal option scrolls the text horizontally. The text automatically scrolls ten characters to the right when the user types a character at the right edge of the edit text boundary. When the user presses Enter, the text scrolls back to the zero position.

If you have associated a database item with the edit control, you will have an additional filtering option available.



► **To filter database items**

On Edit Control Properties

1. Click Filter.



2. Choose one of the Edit Control Filter options.

3. Turn on the Wildcard Display if desired. The Wildcard Display will display an asterisk.

As you add controls to the form, you can tell the runtime engine how to filter the incoming records from the database. For example, if you have two controls on your Find/Browse form that you want to filter on, the resulting SQL statement generated will be an “AND” condition for each condition. For example, if you have Search Type and Alpha Description as the controls on the form, the filter criteria for Search Description should be “>=” and the Search Type control should be “=”. If a user types in “D*” and puts a “V” in the Search Type field the resulting SQL statement looks like the following:

```
SELECT * FROM F0101 WHERE (ABAT1 = “V” AND ABDC LIKE “D%”)
ORDER BY ABAN8 ASC
```

Another situation where you might want to use filter fields is when records need to fall between two values. In this case you use a Range Filter. For example, in distribution a status is assigned to each line of the Order. One status is the current status and the other status is the Next Status. In this example you would filter records greater than or equal to the Present Status and less than or equal to the Next Status. You would drop one, filter, and then drop the next one.

Creating UDC Edit Controls

Use a User-Defined Code (UDC) Edit Control for a field that accepts only specific values defined in a UDC table. Associate a UDC edit control with a database item or data dictionary item.

The visual assist flashlight automatically appears adjacent to the UDC edit control field. When you click the visual assist flashlight, the attached Search and Select form displays valid values for the field.

To create a UDC edit control, you must:

- Associate the data item with a specific UDC table in the data dictionary.
- Create a search and select form for displaying valid values from the UDC table.

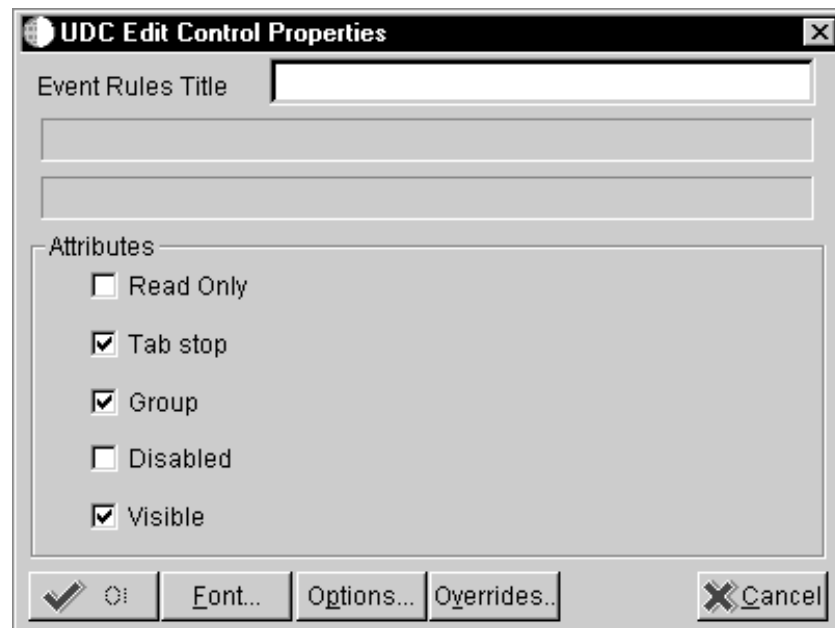


To create a UDC edit control

On the form with which you are working

1. From the Insert menu, choose UDC Edit.
2. Position the control outline on the form and click.
3. To define the UDC edit control properties do one of the following:

- Double-click the UDC edit control
- Choose the UDC edit control and choose Properties from the Settings menu.



4. Complete the following field:
 - Event Rules Title
5. Click any of the following options to alter the behavior of this control:
 - Read Only
 - Tab Stop
 - Group
 - Disabled
 - Visible
6. Click one of the following to further edit the control:
 - Font
 - Options
 - Overrides

See Also

- *Associating Database Items, Dictionary Items, or Descriptions*

Disconnecting Static Text from Controls

You can disconnect the static text from an edit control or a UDC edit control so you can move the static text and edit/UDC control separately or to delete either the text or the control.

If you disconnect the static text of the data item for the edit box, and then delete the static text from the form it will have the same effect as associating an edit control with a database item.

In some cases, the text or description may not be necessary. For example, in Purchase Order Entry, the order type follows the order number, but the description “Order Type” is not necessary.



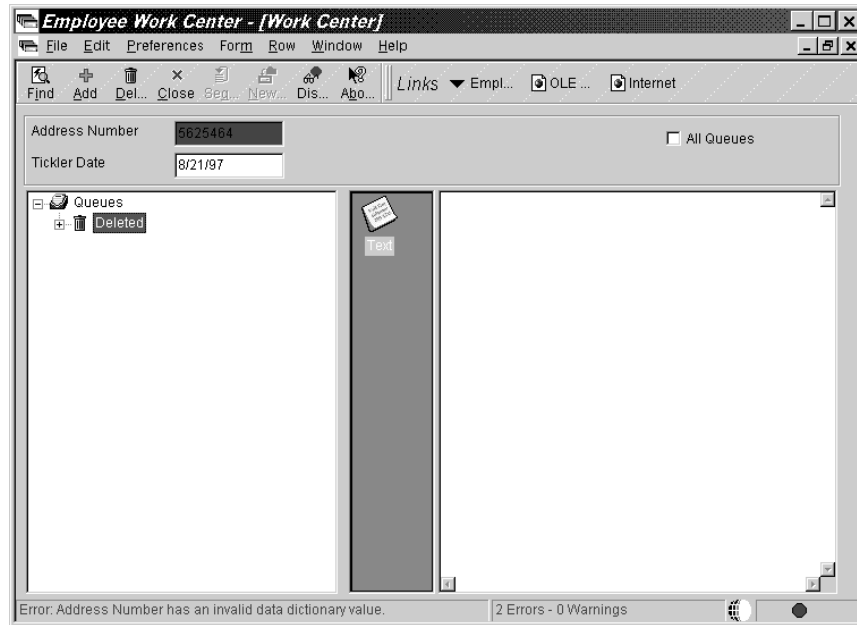
To disconnect static text from an edit control or UDC control

On the form with which you are working

1. Choose the item for which you want to disconnect the static text from the control.
2. From the Edit menu choose Disconnect.
3. Now you can move the static text and edit/UDC box separately. You can use the Cut or Clear option to delete either one.

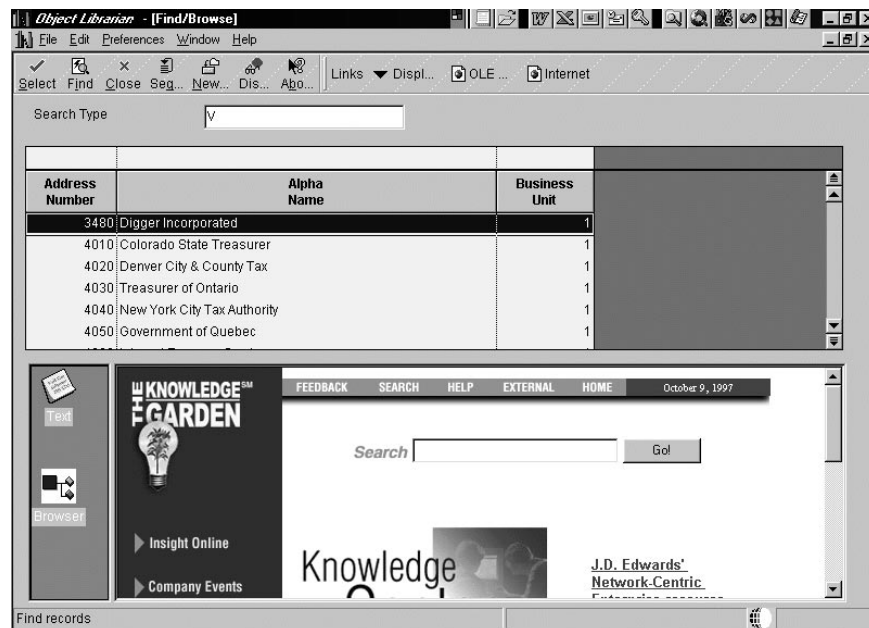
Creating Media Object Controls

The media object control is a very specialized control. You can use this control to allow the user to enter text or attach objects. You can place this control on any form type except the message box. The example below displays an application where J.D. Edwards uses the media object control.

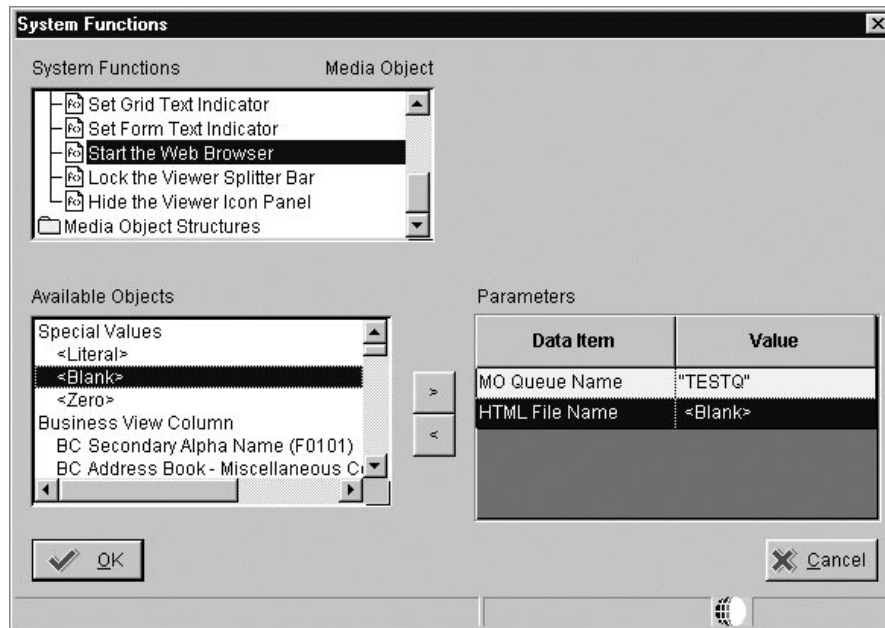


You can use the media object control in a variety of ways. You can add images, shortcuts to other applications, and text.

For example, if you want to display a certain file available on the Internet, you can attach the media object control to the form and have a link to the Internet.

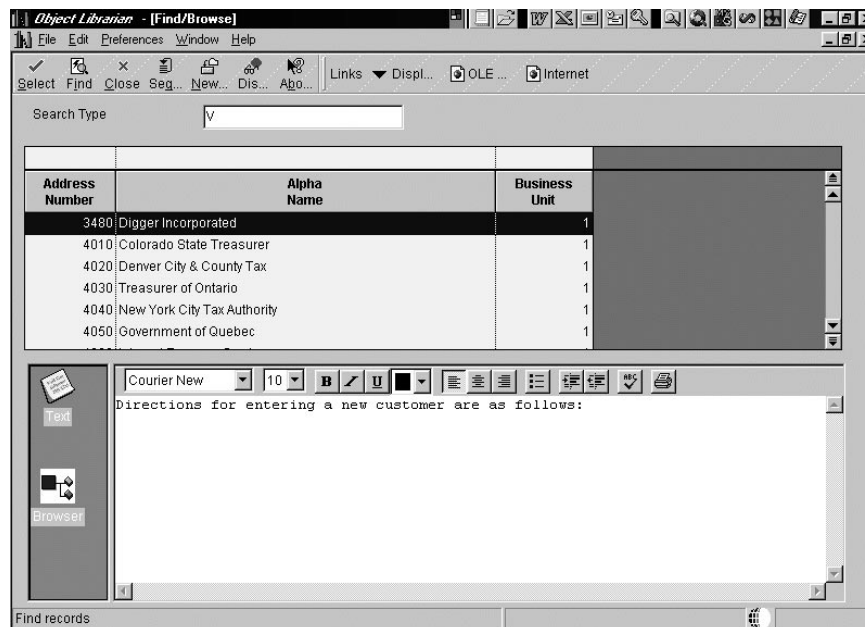


After you define your media object queue for the Internet (for example including a valid HTTP address), you can use the Start Web Browser system function to open the control with the Internet file displayed.

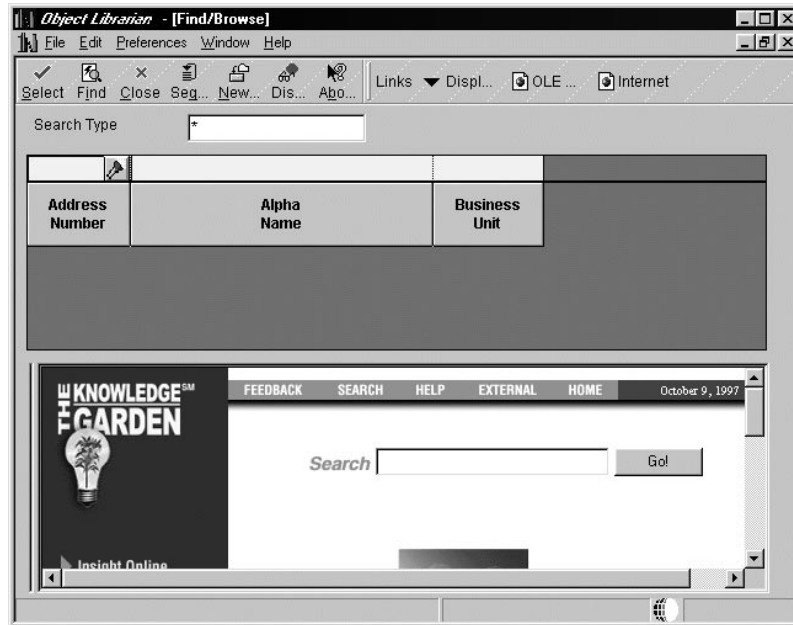


A specific example of how you might use this is if you needed to check a shipping vendor's Web page to track the status of shipments. You can look up the shipment directly within the media object control.

You can also use the Text feature of the media object control. For example, you can use the media object control to display instructions specific to a particular form.



You can use the Hide Splitter Bar system function if you do not want your employees to have access to other media object functions.



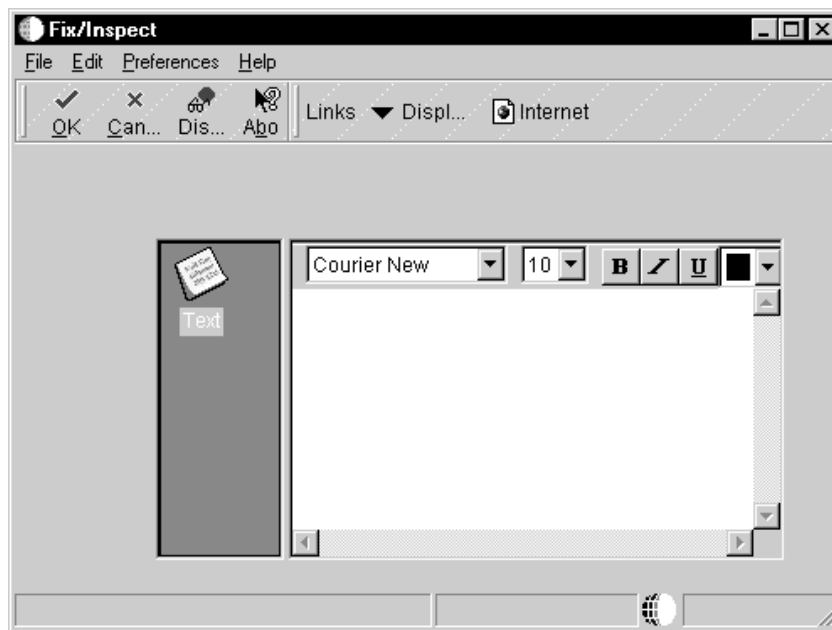
Another way you can use the media object control is to display employee queues where messages are sent. A media object control is laid next to the tree control on a Parent/Child form. Next, event rules are used to establish a relationship between the tree side and the media object side. For example if a message is highlighted in the tree, then the corresponding message text is displayed in the control.

► To create a media object control

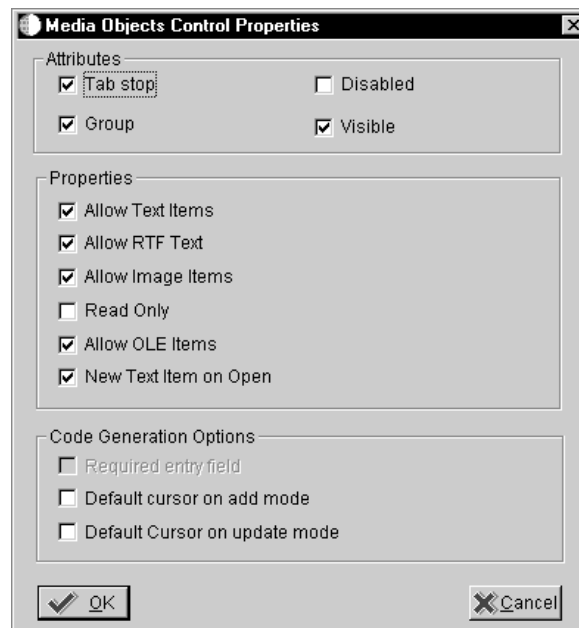
On the form with which you are working

1. From the Insert menu, choose Media Object.
2. Position the control outline on the form and click.

Be sure to size the control so it is large enough to show the entire control. The control looks like the following example when it is sized. If you do not make the control large enough, it will not show everything that is part of the control.



3. To define the media object control properties do one of the following:
 - Double-click the media object control.
 - Choose the media object control and choose Properties from the Edit menu.



4. On Media Objects Control Properties, click one or more of the following Attributes:
 - Tab Stop
 - Group

- Disabled
 - Visible
5. Click one or more of the following Properties:
- Allow Text Items
 - Allow RTF Text
 - Allow Image Items
 - Read Only
 - Allow OLE Items
 - New Text Item on Open
6. Click one or more of the following Code Generation Options:
- Required entry field
 - Default cursor on add mode
 - Default cursor on update mode

Field	Explanation
Tab Stop	Allows the user to tab into this control.
Group	Defines this control as the first in a series of related controls. A group is defined as this control plus all subsequent controls without the Group option turned on. The next control with Group turned on signals the start of a new group (a single control can form a group). Within a group of more than one control, use the arrow keys to move from one control to another instead of pressing Tab.
Disabled	Disables a control. A disabled control is grayed out at runtime.
Visible	Makes a control visible.
Allow Text Items	Allows text items to be added to the Media Object control.
Allow RTF Text	Allows Rich Text Format text to be added to the Media Object control.
Allow Image Items	Allows images to be placed on the Media Object control.
Read Only	Prevents a user from changing the value in a field.
Allow OLE Items	Allows OLE objects to be placed on the Media Object control.
New Text Item on Open	Adds a text item to the Media Object control when the control is first opened.
Required entry field	Prevents the user from leaving this field blank.

Field	Explanation
Default cursor on add mode	Designates this field as the initial position of the cursor in Add mode.
Default cursor on update mode	Designates this field as the initial position of the cursor in Update Mode.

Creating Bitmap Controls

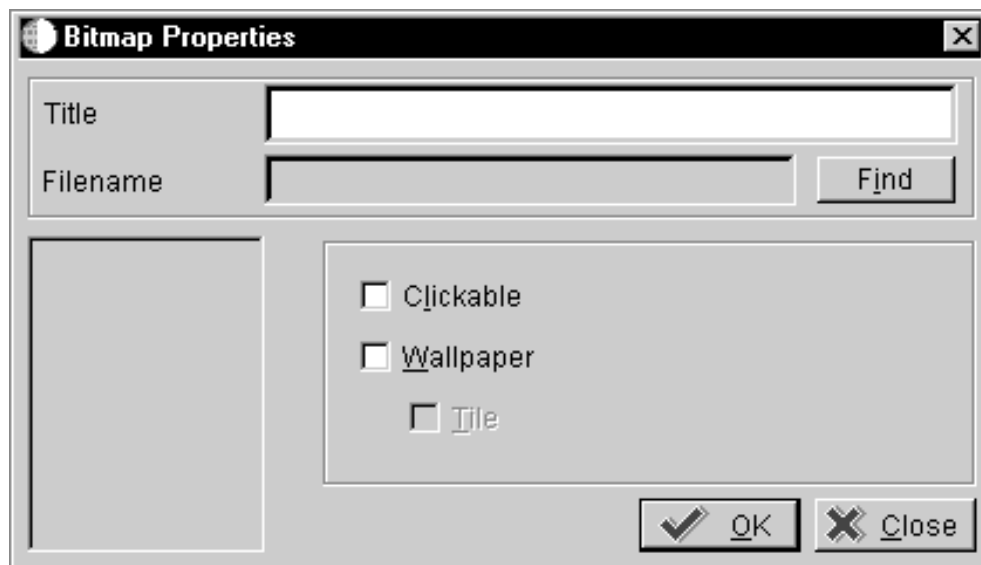
You can create a control that looks like a picture or other art by using a bitmap control. You can also use the bitmap control to create background wallpaper for your form.

► To create a bitmap control

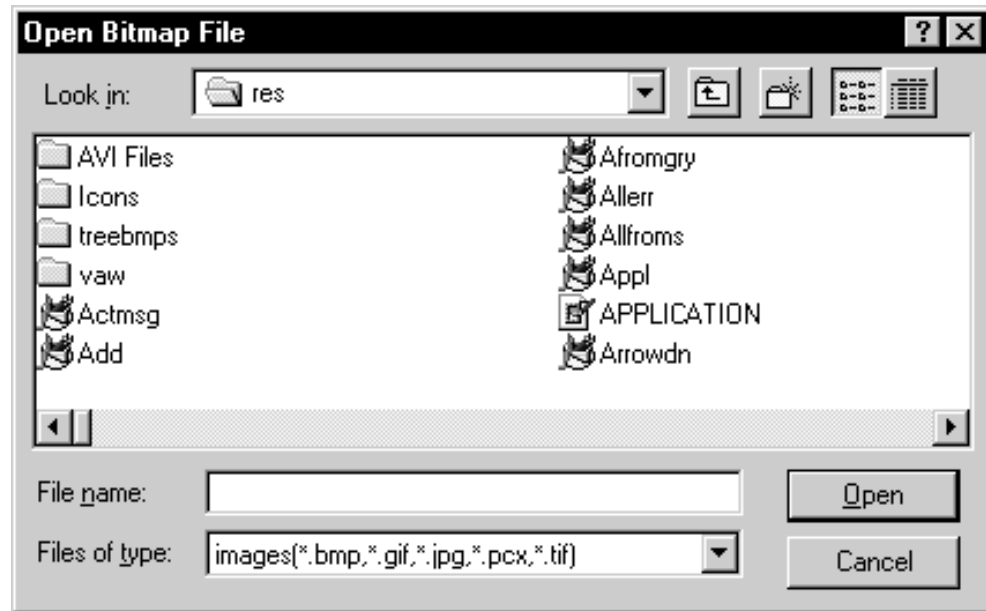
On the form with which you are working

1. From the Insert menu, choose Bitmap.
2. Position the control outline on the form and click.

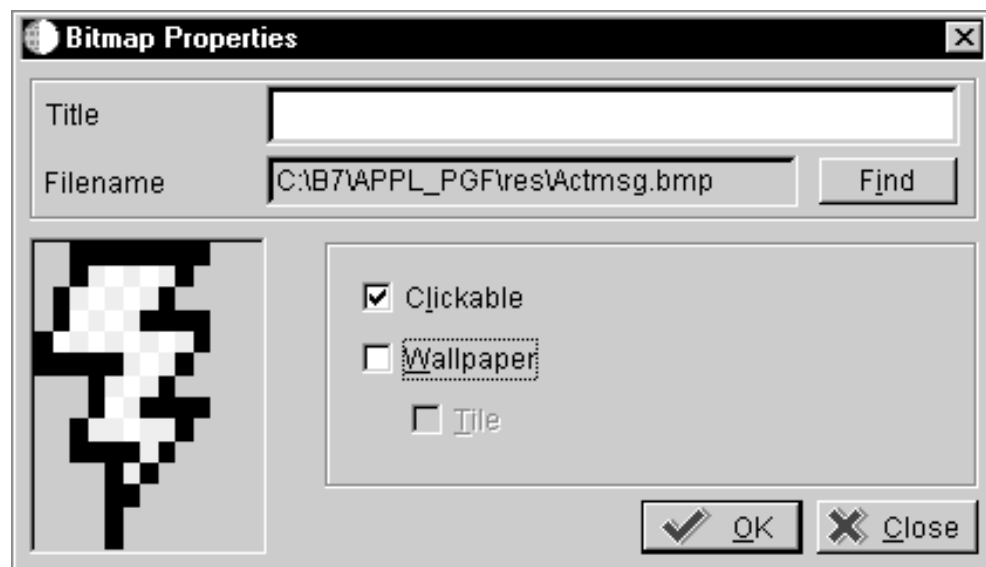
The Bitmap Properties form appears.



3. Click Find to find the file you wish to use for your bitmap control



The Bitmap Properties form displays the bitmap you have chosen.



4. Click one or more of the following attributes:

- Clickable

If you turn on the Clickable option, a Button Clicked event is enabled for the bitmap.

- Wallpaper

If you turn on the Wallpaper option, the bitmap is used as background wallpaper for the form.

- Tile

If you turn on the Tile option, multiple smaller images of the bitmap are used to fill the background instead of one larger image.

Field	Explanation
Clickable	Enables the button to perform an action when it is clicked. If you check this option you must have a Button Clicked event enabled for the bitmap.
Wallpaper	Specifies the wallpaper that is displayed.
Title	The title for the form, field, control, or button. • For a form: This will appear on the form at runtime. • For a field, control, or button: If the title is longer than the size of the control or button, it will be truncated. Resize the control to show the complete title.

Creating a Tree Control

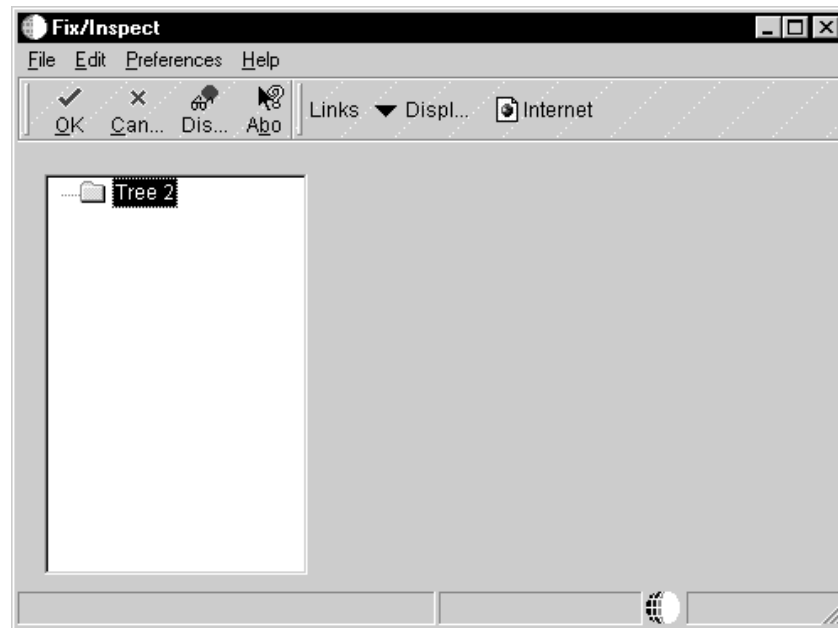
You can use the tree control on any form type. Since the tree control is not limited to a particular business view and does not have any default database functionality, you have more flexibility loading tree node data. There are also tree control system functions available to further customize tree control functionality.

To create a tree control

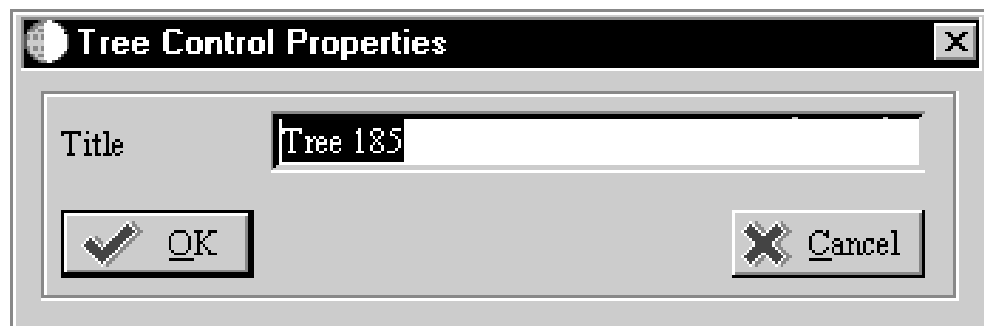
On the form with which you are working

1. From the Insert menu, choose Tree Control.
2. Position the control outline on the form and click.

Size the control if needed.



3. Click the right mouse button to display the Tree Control Properties form and name your control.



You can use Tree Control system functions to add logic to your control. You can use these system functions for actions like contracting and expanding nodes or setting bitmaps for nodes. Refer to the *Online APIs* for more information about these system functions.

Field	Explanation
Title	The title for the form, field, control, or button. • For a form: This will appear on the form at runtime. • For a field, control, or button: If the title is longer than the size of the control or button, it will be truncated. Resize the control to show the complete title.
Tree Folders	

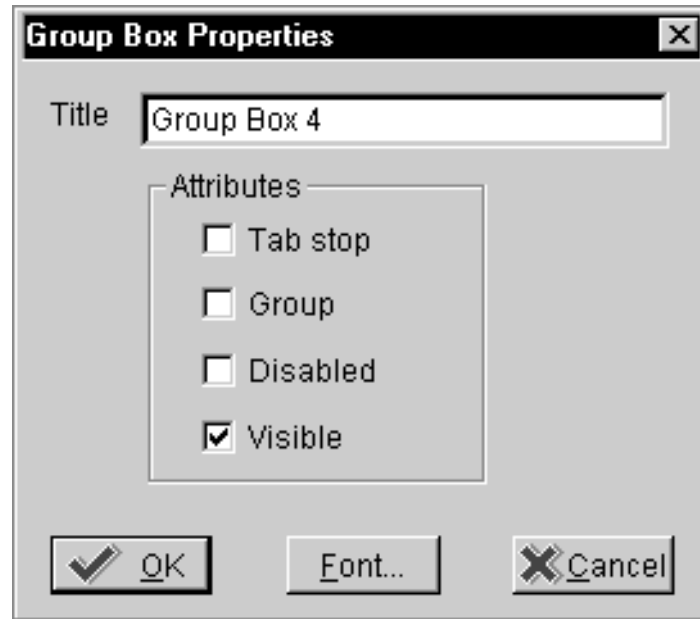
Creating Group Boxes

A group box visually categorizes and defines a group of fields by surrounding the controls and providing an optional title to the group.

To create a group box

On the form with which you are working

1. Choose the controls to be included in the group box by drawing a box around the fields.
2. From the Insert menu, choose Group Box.
3. To define the group box properties do one of the following:
 - Double-click the group box
 - Choose the group box and choose Properties from the Settings menu.



4. Complete the following field:
 - Title
5. Click one or more of the following Attribute selections:
 - Tab Stop
 - Group
 - Disabled
 - Visible

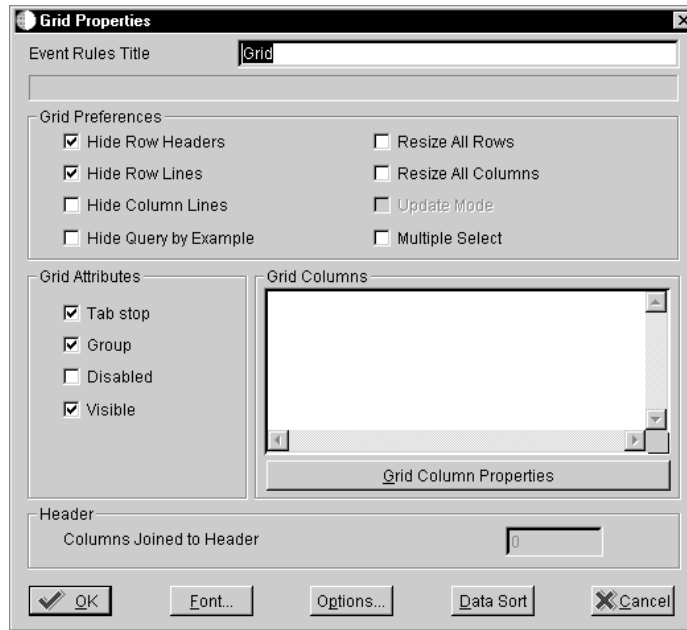
Defining Grid Properties

You can define grid properties such as the font or sort order for a grid.

To define grid properties

On the form with which you are working

1. To define the grid control properties do one of the following:
 - Double-click on the grid
 - Choose the grid control and choose Item Properties from the Edit menu.



2. Click any of the following Grid Preferences:

- Hide Row Headers

The row header must be displayed when media objects are used. This is the gray column to the left of the grid row where the paper clip icon displays.

- Hide Row Lines

This is standard for Find/Browse forms.

- Hide Column Lines
- Hide Query By Example
- Resize All Rows
- Resize All Columns
- Update Mode
- Multiple Select

This option must be turned on in order to make a form multiselect, for example to allow multiple grid rows to be selected for an operation. The property event rule for the button (for example Select or Delete) must have the Repeat Event Rule for Grid option turned on.

3. Click any of the following Attributes:

- Tab Stop
- Group

- Disabled
 - Visible
4. To define properties for a selected column, choose an item from the Grid Columns list and click Select.
 5. Click any of the following to specify additional grid properties:
 - Font
 - Options

For information about grid options refer to *Defining Options For a Grid*.

- Data Sort

If you click Data Sort, the following form appears and you can choose the way you want your data sorted.

Choose an Available Column and click the right arrow button to sort on that column. After the column appears in Columns Sorted, choose it and right mouse click on the A/D (ascending/descending) column to toggle to either an ascending or descending sort order for that column. The columns are sorted starting with the top one under Columns Sorted. You can sort on several columns if needed. Use the arrows in the menu to change the order of your sorted columns.

Field	Explanation
Hide Row Numbers	Hides the horizontal row headings.

Field	Explanation
Hide Row Lines	Hides the horizontal row lines in the grid.
Hide Column Lines	Hides the vertical column lines in the grid.
Hide Query By Example	Suppresses the QBE functionality by hiding the QBE line. Do this only if the grid does not contain any fields from the table, that is, there is nothing on which to query.
Resize All Rows	Enables the user to enlarge the grid rows.
Resize All Columns	Enables the user to widen the grid columns.
Update Mode	Defines the grid to be input capable. This only works on the header/detail and headerless detail forms. This option has no effect on the find/browse form because that form type is incapable of input.
Multiple Select	Allows you to select more than one row in the grid. When you click on Select, Event Rule logic is performed on the first selected row, then the second selected row, and so on. There is no default behavior on Select.
Grid Column List	Displays columns in the grid.
Select Grid Column	Displays grid column properties for a selected column.
Number of columns joined to	Used for backwards compatibility. All new forms should have a zero in this field.

See Also

- The *Foundation* guide for how to customize the grid at runtime.

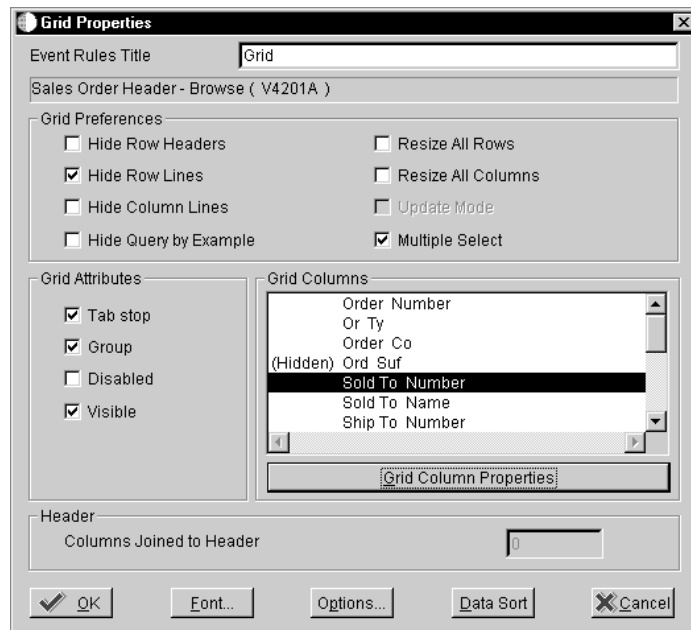
Defining Grid Column Properties

You can set properties for each grid column. On Find/Browse forms, you cannot attach event rules to grid columns.

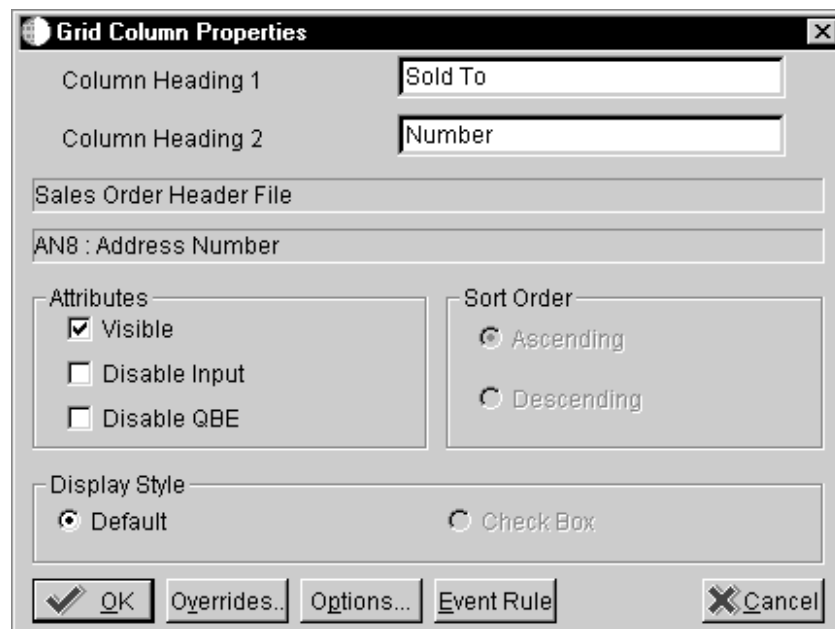


To define grid column properties

On Grid Properties



1. Choose the grid column for which you wish to define properties.
2. Click the Grid Column Properties button.



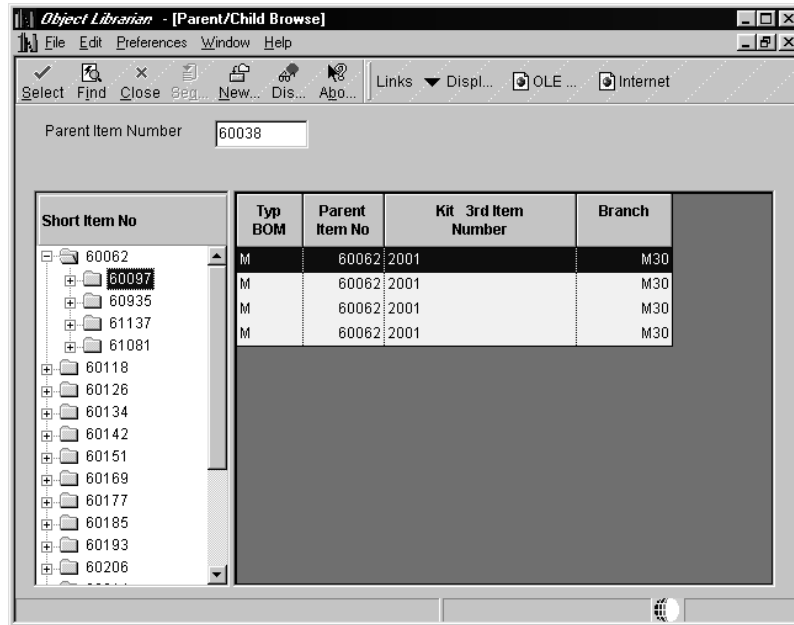
3. Complete the following fields:
 - Column Heading 1
 - Column Heading 2
4. Click one or more of the following attributes:
 - Visible

- Disable input
 - Disable QBE
5. Click one of the following options to indicate the sort order:
 - Ascending
 - Descending
 6. Click one of the following display styles:
 - Default
 - Check Box (Check Box is currently unavailable)
 7. Click the following to define additional grid properties:
 - Overrides
 - Options
 - Event Rules

Field	Explanation
Column Heading 1/Column Heading 2	Specifies the text for the first and second lines of the column heading.
Visible	Makes a control visible.
Disabled	Disables a control. A disabled control is grayed out at runtime.
Disable QBE	Prevents a query for the grid column.
Ascending/Descending	Indicates the ascending or descending sort order for this column.
Display Style-Default or Check Box	Designates whether the data item is displayed only in the grid or available as a check box.

Defining Properties for Parent/Child Controls

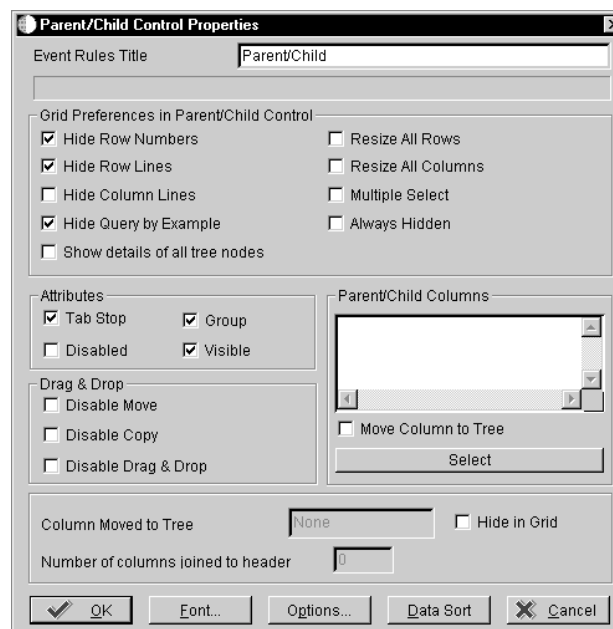
The Parent/Child form has a special parent/child control that automatically appears with the form. The control is a composite control with a tree control on the left side and a grid control on the right. The tree and grid control portions are both tied to the same business view. This control is only available on the Parent/Child form. Refer to *Understanding Form Types* for more information about the Parent/Child form. The following graphic shows a Parent/Child form.



► **To define properties for the parent/child control**

On the form with which you are working

1. Do one of the following:
 - Double-click on the grid portion of the parent/child control
 - Right-click the parent/child control and choose Properties.



2. Click any of the following Grid Preferences:
 - Hide Row Numbers
 - Hide Row Lines
 - Hide Column Lines
 - Hide Query By Example
 - Show details of all tree nodes
 - Resize All Rows
 - Resize All Columns
 - Multiple Select
 - Always Hidden
3. Click any of the following Attributes:
 - Tab Stop
 - Disabled
 - Group
 - Visible
4. Click any of the following Drag & Drop options:
 - Disable Move
 - Disable Copy
 - Disable Drag & Drop
5. To define properties for a selected column, choose an item from the Parent/Child columns list and click Select.
6. Complete the following field:
 - Number of columns joined to
7. Click any of the following buttons to specify additional properties:
 - Font
 - Options
 - Data Sort

You can also double click on the grid to display the properties for the grid. Choose the grid column you wish to display in the tree within the scroll window and select the check box to move it to the tree side of the control. This hides the chosen column from the right side of the grid (default is to hide unless checkbox is unchecked) and displays it only in the left side of the control or the tree. This also moves the column description over.

You can select multiple objects; however, they must be at the same node level. There are two modes available. In the first mode, there is a one-to-one correspondence

between the tree and the grid. The other mode shows all details of the tree. You can also load trees pre expanded. This option is on the right mouse menu.

After the above steps you will need to set up either a parent/child relationship or use event rules to load child nodes to the tree. The method used will depend on whether your table has an inherent parent/child relationship. In the following examples the first example describes a situation where there is an inherent parent/child relationship in a table. The second example describes a situation where the data does not have an inherent parent/child relationship by keys, but can be displayed in a hierarchical manner for readability.

Field	Explanation
Event Rules Title	The default name of the control for use in an event rule. The event rules title defaults to the static text associated with an item. If there is no associated static text and you want to use this control in an event rule, enter a name.
Hide Row Numbers	Hides the horizontal row headings.
Hide Row Lines	Hides the horizontal row lines in the grid.
Hide Column Lines	Hides the vertical column lines in the grid.
Hide Query By Example	Suppresses the QBE functionality. Do this only if the grid does not contain any fields from the table, that is, there is nothing on which to query.
Show details of all tree nodes	
Resize All Rows	Enables the user to enlarge the grid rows.
Resize All Columns	Enables the user to widen the grid columns.
Multiple Select	Allows you to select more than one row in the grid. When you click on Select, Event Rule logic is performed on the first selected row, then the second selected row, and so on. There is no default behavior on Select.
Always Hidden	Hides the grid control on the Parent/Child Form.
Disable Move	Disables the Move Here option on the popup menu at the end of a drag and drop using the right mouse button.
Disable Copy	Disables the Copy Here option on the popup menu at the end of a drag and drop using the right mouse button.
Disable Drag and Drop	Prevents a user from performing a drag and drop operation in the tree view of the control.
Parent/Child Columns	
Move Column to Tree	Displays the text for a column as the text of the corresponding tree node.

Field	Explanation
Hide in Grid	Within the Parent/Child Properties form, this option determines which columns in the related business view you want to hide or display in the detail area. When you designate the Move Column to Tree option, the Hide in Grid option is automatically selected. The default is to display all columns.
Number of Columns Joined to Header	Used for backward compatibility. All new forms should have a zero in this field.

Example: Inherent Parent/Child Relationship

This example describes a case in which there is an inherent parent/child relationship in a table. For instance, in the Bill of Materials table (F3002), IXKIT is the short item number for the kit. Kits are composed of multiple items within IXKIT, represented by IXITM, the short item number for the item.

Kit item 60038 includes items 60062, 60118, 60126, and other items as its children.

<i>IXTBM</i>	<i>IXKIT (Parent)</i>	<i>IXITM (Child)</i>	<i>IXKITA</i>	<i>IXMMCU</i>	<i>IXAITM</i>
<i>M</i>	60038	60062	220	<i>M30</i>	2001
<i>M</i>	60038	60118	220	<i>M30</i>	2006
<i>M</i>	60038	60126	220	<i>M30</i>	2007

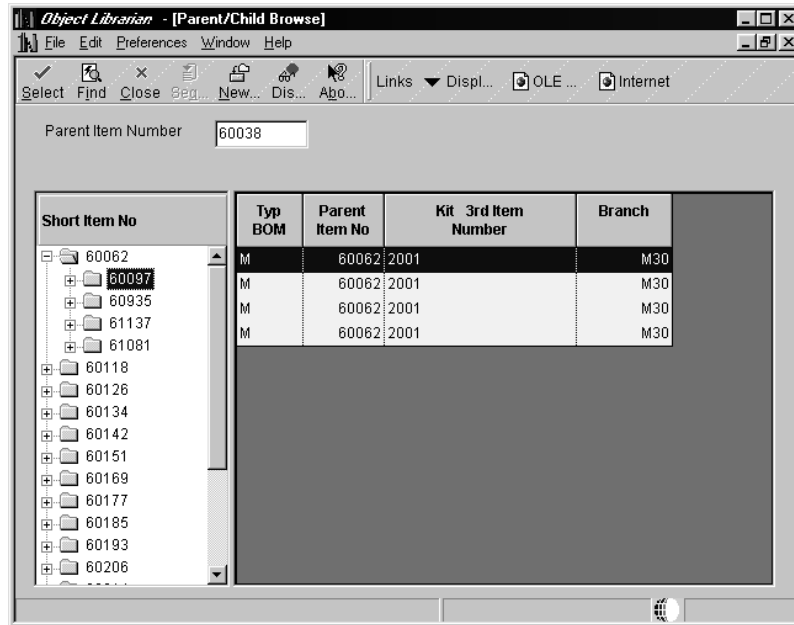
Each one of the items (IXITM) can be a kit within itself.

Item 60062 includes 60097, 60935, 61137, and other items as its children.

<i>IXTBM</i>	<i>IXKIT (Parent)</i>	<i>IXITM (Child)</i>	<i>IXKITA</i>	<i>IXMMCU</i>	<i>IXAITM</i>
<i>M</i>	60062	60097	2001	<i>M30</i>	2004
<i>M</i>	60062	60935	2001	<i>M30</i>	9011
<i>M</i>	60062	61137	2001	<i>M30</i>	9031

Because this inherent relationship exists within the table, you can develop a parent/child application over the Bill of Materials table F3002 (using V3002A business view).

The following form represents this data.



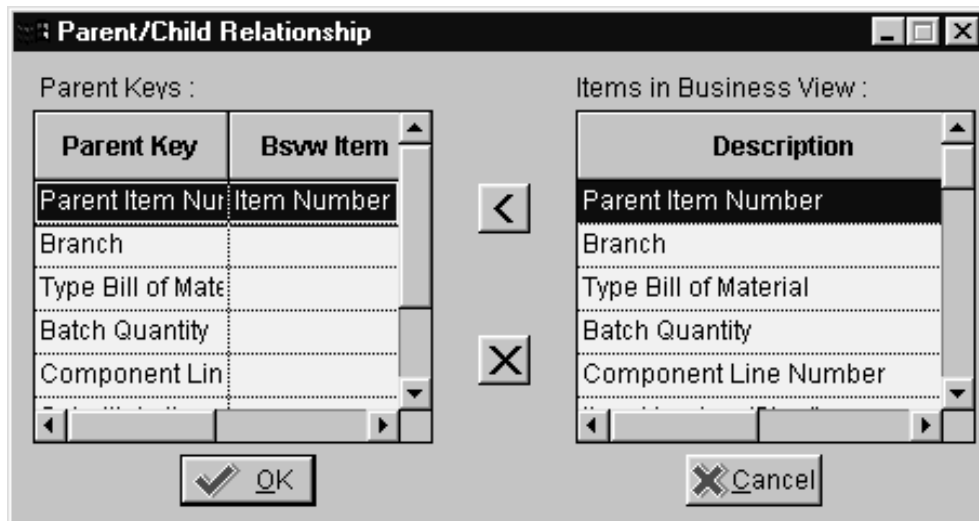
This structure represents data from the same table shown in the parent/child format. When you first inquire on the 60038 item in the filter field you generate the same SQL statement as you would on a normal Find/Browse form.

When you click on the + node next to item 60062, the runtime engine takes the value of the Item Number and moves it into the Parent Item Number, making it the parent for the fetch.

To set up the parent/child relationship to be used when the runtime engine loads nodes to the tree:

- Focus on the parent/child control.
- From the Form menu, choose Parent/Child Relation.

The following form appears:



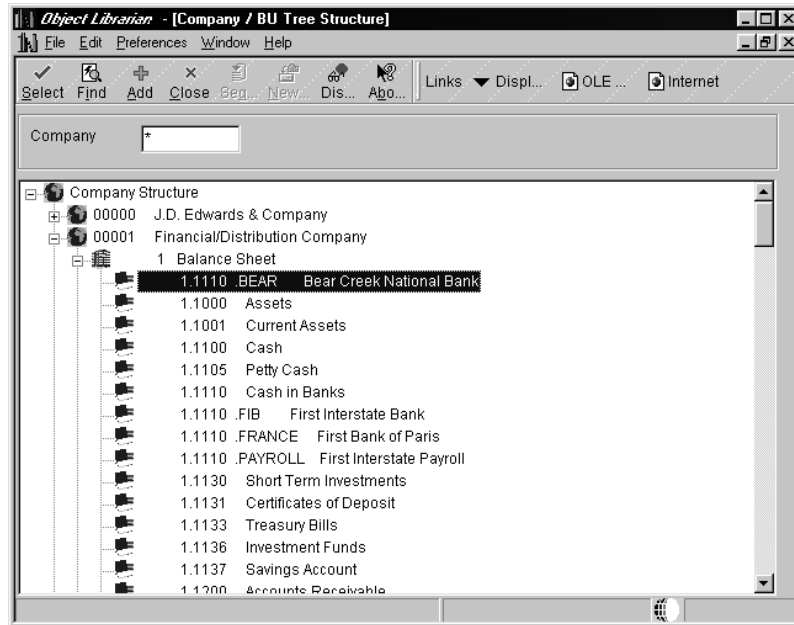
- Indicate which child item should be used as the parent for loading the next level nodes.

A Parent/Child form is an extension of the Find/Browse form, where the fetches are done by the tool using different keys to access its data. Some things to consider when you create a Parent/Child form include:

- You usually use the parent field for the filter.
- You usually use the child for the left side of the parent/child control.
- The data item type between the parent and the child must be compatible.
- You must establish the parent/child relationship correctly. You usually move the child item into the parent field.
- Subsequent (RESET) fetches are done by the tool only when a node is clicked. The tool takes the value of the child at that node and substitutes it into the parent field for the fetch. Once a fetch is performed for that node, that information is saved in a runtime structure. If the same node is clicked again, the data will not be fetched from the database, but instead from the runtime structure.

Example: No Inherent Parent/Child Relationship

This example describes a situation where there is no inherent parent/child relationship. In this situation, the runtime engine does not automatically fetch records because the parent/child relationship is not inherent, and the relationship is between data items of different types. In this case, the data item you chose to be in the left side of the parent/child control is usually a string (alphanumeric) variable that will hold any data type.



The parent/child relationship is not inherent to a particular table because the application is accessing multiple tables, such as Company Master, Account Master, and Chart of Accounts, to get information.

Page-at-a-Time Processing

Page-at-a-time processing ensures that each fetch only fetches one page of data. Page-at-a-time processing is the default mode for all parent/child forms.

During page-at-a-time processing in standard mode, the page size is defined as the number of nodes that can fit in the current view. When the Find process starts only one page of the first level nodes is fetched from the table and inserted in the tree. When the user scrolls down a new page of data is fetched.

Page-at-a-time processing for expand-all style Parent/Child forms is similar to that for standard mode. When the Find process starts, one page of first level nodes are fetched from the table and inserted into the tree. Then all of the first level nodes are expanded. Each expansion fetches only one page of data. Because the tree expands exponentially in expand-all mode, a tree that is very deep may impact performance.

Tree Nodes

When the parent and child nodes come from different tables or are of different data types, the parent/child relationship is not automatically set up. When this is the case the runtime engine does not automatically fetch the child database records because it does not know which table to retrieve them from.

Parent Nodes

If possible, use the runtime engine to load the initial set of parent nodes to the tree for you. You do this by using the based-on view, which is a view over the upper most node's table. This way a parent filter can still be used in the control, and the runtime engine will load the first level nodes to the tree. If you cannot do this, you must insert the first level nodes yourself. To do this, you usually use Table I/O on the *Button is Clicked* event of the Find button. You use the same methods as you do to insert child nodes. Use a *Suppress Find* system function to stop the runtime engine from attempting to load any nodes.

Child Nodes

Whenever a node is expanded, the system function *Suppress Fetch on Node Expand* is called from the event *Tree Node Is Expanded*. This tells the runtime engine not to do any fetches because event rules will handle the loading of the child nodes. *Tree Node Is Expanded* is the main event of the application. This event occurs when the tree node is expanding (for example, the plus next to a child node is expanded for the first time). You place event rules on this event to read the next records to be loaded to the tree as children of the expanded node. You can use table I/O or business functions to retrieve these records. Often the children may be coming from different tables based on the type of parent node that is expanded. If possible, you should do a SELECT and then use the FETCH NEXT command to retrieve records in a DO WHILE loop. The GB runtime data structure should be populated with data from the records read in the loop, and then an INSERT GRID BUFFER ROW system function should be called. This parent child system function is different than the INSERT GRID BUFFER ROW in the normal GRID section. At this point you also have the ability to set your custom tree bitmaps using the SET TREE NODE BITMAP system function.

Example

In the following example, event rules are attached to an application on the *Tree Node is Expanding* event:

```

Suppress Fetch on Node Expand(FC Parent/Child)
//
// Here are the variables to get out of the account and the business unit
loop
// being initialized.
VA frm_OutOfLoop = "0"
VA frm_ExistAcctLoop = "0"
//
// If Loop looking at the GC Business Unit Field.
If GC BusinessUnit is equal to <Blank>
//
// Select the F0006 differently if there is one company or all companies
//
F0006.Open
If VA frm_AllCompanies is equal to "1"
VA frm_CurCompany = GC Co
F0006.Select
Else
VA frm_CurCompany = BC Company
F0006.Select
End If
//
// While Loop which fetches all the all the business unit for a specific
company.
// If the company changes we get out of the loop.
While VA frm_OutOfLoop is equal to <Zero>
// Fetch the records from the F0006 Table.
F0006.FetchNext
GB BusinessUnit = GB Companies
If SV File_IO_Status is equal to CO SUCCESS

GB Co = BC Company
VA frm_PreCompany = BC Company
VA frm_ConcateBuDesc = " "
VA frm_ConcateBuDesc = concat([VA frm_ConcateBuDesc],[VA frm_NameOfBU])
GB CompanyStructure = concat([GB Companies],[VA frm_ConcateBuDesc])
GB Companies = concat([GB Companies],[VA frm_ConcateBuDesc])
//
// Tells the Level of the Tree Structure.
GB LevelOfTreeInt = "1"
//
Insert Grid Buffer Row(FC Parent/Child, <After Last Row>, <Yes>, <No>, <No>,
<No>, <No>, <Yes>)
Set Tree Node Bitmap(FC Parent/Child, <Last Grid Row>, BussUnit.bmp, <Yes>)
//
//
If VA frm_PreCompany is not equal to VA frm_CurCompany
VA frm_OutOfLoop = "1"
End If
Else
VA frm_OutOfLoop = "1"
End If
End While
F0006.Close
Else
// Loop thru the F0901 to get the corresponding accounts.
//
VA frm_CURBU = GC BusinessUnit
F0901.Open
F0901.Select
If SV File_IO_Status is equal to CO ERROR

//
End If

```

```
GC Companies = " "  
GB Companies = " "  
Business Unit, Object, Subsidiary Merge  
GB Companies = concat([VA frm_DBANI],[VA frm_AcctDesc])  
GB CompanyStructure = concat([VA frm_DBANI],[VA frm_AcctDesc])  
GC Companies = VA frm_AIDF0901  
//  
// Tells the level of the Tree Structure '2'  
GB LevelOfTreeInt = "2"  
//  
Insert Grid Buffer Row(FC Parent/Child, <After Last Row>, <Yes>, <No>,  
<No>, <No>, <No>, <No>)  
Set Tree Node Bitmap(FC Parent/Child, <Last Grid Row>, accounts.bmp, <Yes>)  
End If  
End While  
End If  
FC BUFrom = " "
```

Dragging and Dropping

When a Parent/Child form is created, both move and copy drag-and-drop operations are enabled. You can turn these options off in the properties for the form. If drag-and-drop is turned off and a user attempts to drag a node, the cursor will indicate that dragging is not allowed. None of the drag events will execute. You can control operations to the database using the drag-and-drop events:

- *Begin Drag* - If drag-and-drop is allowed then on the Begin Drag Operation event, GCs are copied to GBs.
- *Drag Over Node* - You can attach event rule logic to this event to validate that the node the dragged record is about to be dropped on is a valid situation. If it is not, you can use a system function to change the cursor to a No Drop Cursor to indicate that dropping the record there will not be allowed. If the cursor is not the no drop cursor, when the record is dropped, the event *End Drag* will run.
- *End Drag* - You can attach event rules to this event to update or insert information that has been moved or copied via the drag. Be aware of the impact from using *Insert Grid Buffer Row* in the *End Drag* event, as well as deleting the grid row dragged if the operation is a Move.
- *Drag Mode* - The Drag Mode system value can be checked at any time to see what kind of drag a user is doing, for example, whether the drag is a move or a copy.

Tree Node is Selected

This event runs every time a tree node is selected, either by clicking on it once with the mouse or by moving an arrow up and down the nodes. You can place event rules that need to run when a mode is selected on the *Tree Node is Selected* event. The Work Center application uses this feature to load the media object that sits next to the tree with the message information for each node. You can also use this when controls or exits must be protected based on the kind of node that is selected.

Defining Options for Forms, Grid, and Edit Controls

You can use options to define additional properties for forms and edit controls or UDC edit controls. After you create the form and controls, you can define options for a:

- Form
- Grid
- Control (edit and UDC edit only)

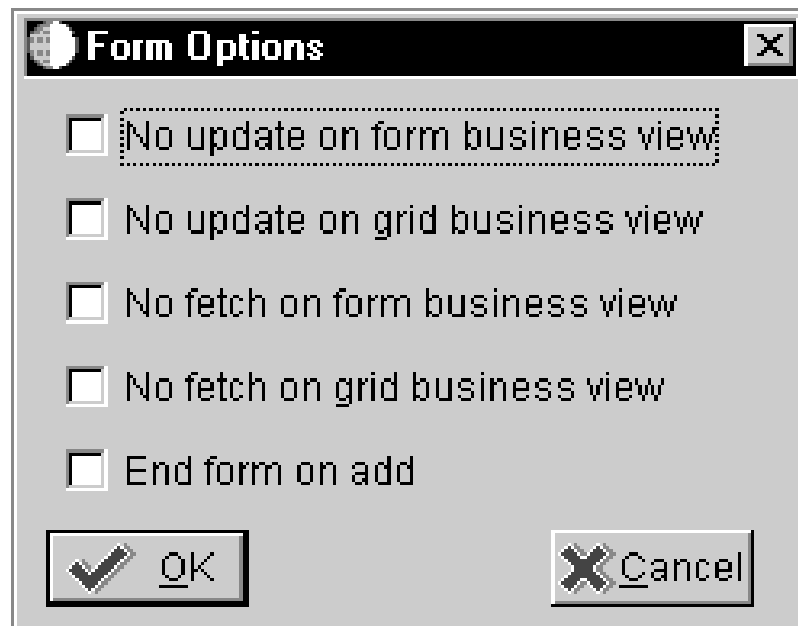
If you change any control options, you do not have to regenerate the application. Control options are interpreted at runtime.

Defining Options for a Form

► To define control options for a form

Focus on the form and choose Properties from the Form menu. On Form Properties

1. Click the Options button.



2. Click any of the following options to override the default processes:
 - No update on business view
 - No update on grid business view
 - No fetch on form business view
 - No fetch on grid business view

- End form on add

Field	Explanation
No update on business view	Ignores updates to tables when you click OK.
No update on grid business view	Ignores updates to the grid when you click OK.
No fetch on form business view	Does not fetch records from the table when you enter a form.
No fetch on grid business view	Does not fetch records from the table when you enter a grid.
End form on add	Returns to the previous form after you add a record and click OK.

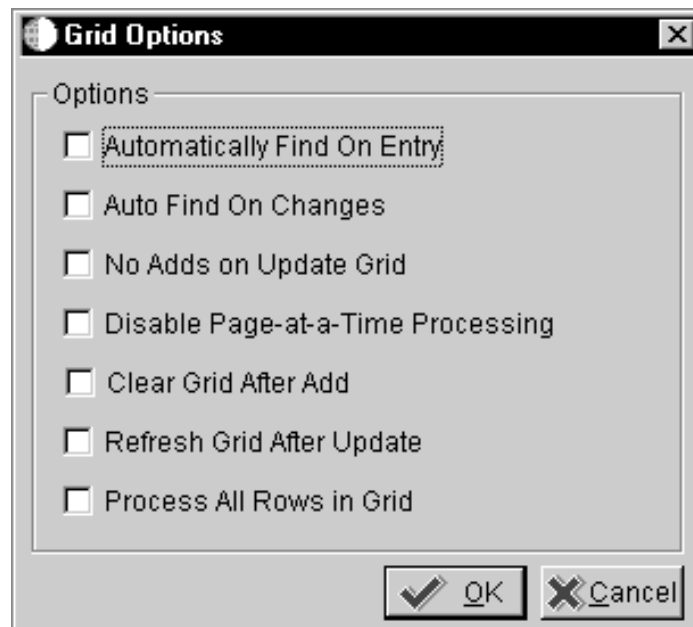
Defining Options For a Grid

► To define grid options

Focus on the grid and choose Properties from the Form menu.

On Grid Control Properties

1. Click the Options button.



2. Click any of the following options to override the default process:

- Automatically Find On Entry
- Auto Find On Changes
- No Adds On Update Grid
- Disable Page-at-a-Time Process
- Clear Grid After Add
- Refresh Grid After Update
- Process All Rows in Grid

Field	Explanation
Automatically Find On Entry	Automatically loads data into the grid upon entering the Find/Browse or Headerless Detail form. Use this feature only if there are not many records to retrieve, otherwise you may slow system performance unnecessarily. Note: J.D. Edwards standard is to turn this option off for Find/Browse forms and to turn it on for Header Detail and Headerless Detail forms.
Auto Find On Changes	Automatically refreshes the grid after a change is made on another form and the user is returned to the Find/Browse form. Because of the impact on system performance, use this option only in applications in which instantaneous, up-to-date information for all users is required. Note: J.D. Edwards standard is to turn this option off.
No Adds On Update Grid	Prevents a user from adding records on an input-capable grid, such as on a Headerless Detail or Header Detail form. The user will still be able to change records.
Disable Page-at-a-Time Process	Page-at-a-time processing loads one page of records into the grid at one time. When the user scrolls past the last record in the page, the next page of records is read into the grid. Turning this option on disables page-at-a-time processing, and OneWorld will load all records into the grid at one time. If there is a large number of records, turning this option on could have a negative impact on performance.
Clear Grid After Add	On Header and Headerless Detail forms, this enables the user to add more records after clicking OK.
Refresh Grid After Update	On Header/Detail and Headerless Detail forms, this reloads the grid from the database in update mode.
Process All Rows in Grid	Processes all grid rows, including those that have not been changed. Use this option carefully because it negatively impacts performance.

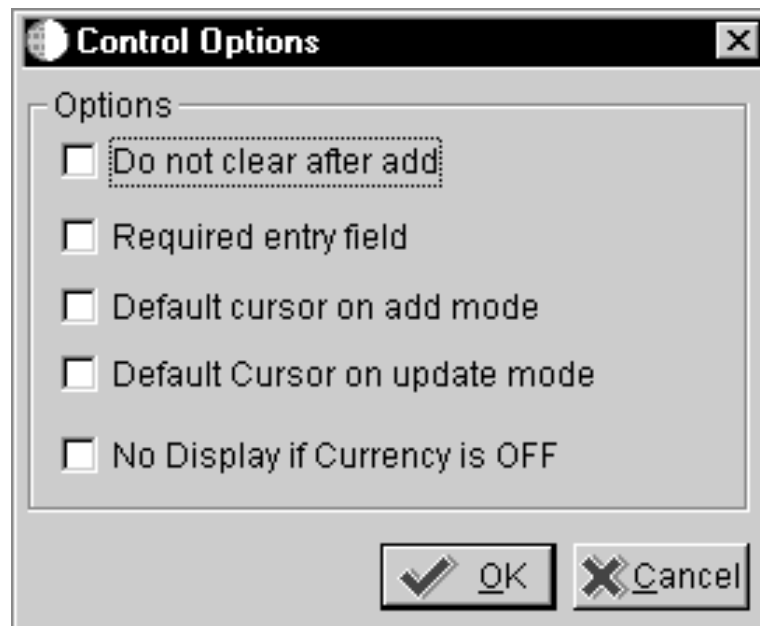
Defining Options For an Edit or UDC Edit Control

► To define control options for an edit or UDC edit control

Focus on the control and choose Properties from the Form menu.

On Edit Control Properties or UDC Edit Control Properties

1. Click the Options button.



2. Click any of the following options:
 - Do not clear after add
 - Required entry field
 - Default cursor on add mode
 - Default cursor on update mode
 - No display if currency is OFF

Field	Explanation
Do not clear after add	Prevents the user from clearing this field after a record is added.
Required entry field	Prevents the user from leaving this field blank.
Default cursor on add mode	Designates this field as the initial position of the cursor in Add mode.

Field	Explanation
Default cursor on update mode	Designates this field as the initial position of the cursor in Update Mode.
No display if currency is OFF	Hides this option if the Multicurrency flag = No in System Setup, General Accounting Constants (P0000).

Associating Database Items, Dictionary Items, or Descriptions

Associate any of the following with a control:

- Database item
- Dictionary item
- Description

Radio buttons, check boxes, edit controls, and UDC edit controls can have associated values. A push button cannot have an associated value. Database items and dictionary items can only be associated with check box or radio button controls. You can look at the properties for a control to see which data dictionary item the control is associated with.

Database Items

If you want the user-entered value for a checkbox or radio button to update the record, then the control must be associated with a database item.

For example, a check box can be associated with the database field, “Taxable.” In the check box properties, the Checked value should be Y and the Unchecked value should be N. These values can be used in event rules and will update the Taxable database field in the table.

Dictionary Items

Because dictionary items never update the table, they can only be used in event rules. The user can enter values in the dictionary item that will be used in event rules or an event rule can produce a value that will be displayed in a dictionary item.

Descriptions

When you associate a description with an edit control or UDC edit control, a description of the value in the control is automatically displayed next to it. Associating a description is optional but is useful as a visual aid on the form.

For example, suppose the address book number 1001 appears in an edit control or in a UDC edit control. When the user tabs out of the control, the address book name “XYZ Company” is displayed next to the control.

To associate an item or description with a control

On the form with which you are working

1. Click on the control.
2. From the Edit menu, click one of the following:
 - Associate Database Item
 - Dictionary Item
 - Description
3. If you are associating a database item or dictionary item, locate and choose the item.
4. If you are working with a description, move the control to the desired position on the form and click once to place it on the form.

If you associate radio buttons and you want them to be mutually exclusive, select them and apply a group box.

Changing Tab Sequence

The order in which the cursor travels through the controls on your form is determined by the tab sequence of the controls.

Each control that is designated as a tab stop is numbered according to the way the cursor travels. For example, when the form is first opened, the cursor will be on the control labeled number one. When the user presses the Tab key, the cursor moves sequentially to the next tab on the form.

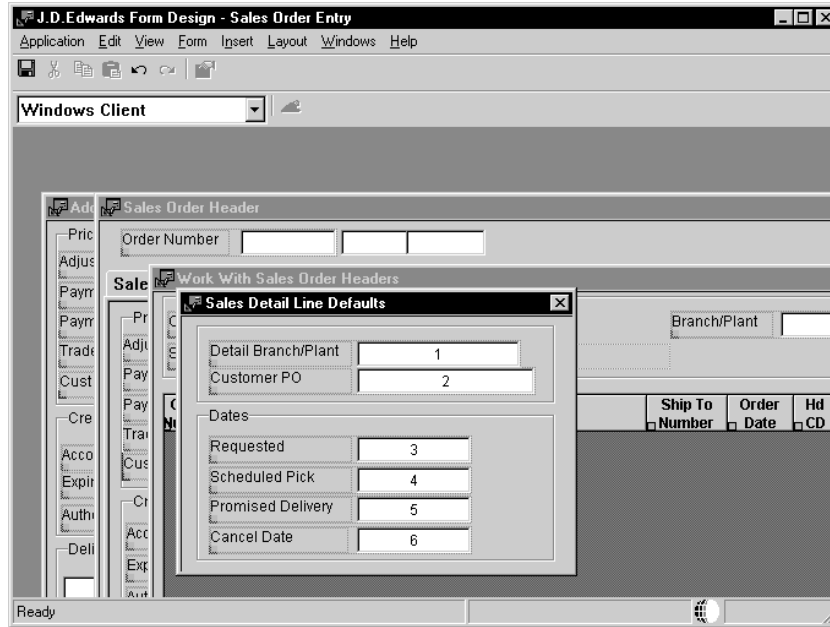
Only those controls that are designated as tab stops in the control properties are affected by the tab sequence.

The default tab sequence is the order in which the controls are placed on the form.

To change the tab sequence

On the form with which you are working

1. From the Form menu, choose Tab Sequence.



The controls are already numbered.

2. Click the controls in the order you want the cursor to travel.

The first control you click becomes number one, the second becomes number two, and so on. To reset the numbers to the original values, click the right mouse button anywhere in the window.

3. To hide the tab sequence, from the Form menu, choose Tab Sequence again.

To set the tab sequence for a group of controls (such as a set of radio buttons), the Tab Stop and Group properties must be turned on for the first control in the group. All other controls in the group should have these properties turned off. This ensures that the cursor tabs to the first control in the group and skips the others. To move within the group, use the arrow keys.

The following two options override the first tab stop:

- Default cursor on add mode
- Default cursor on update mode

For more information on using these two options, see *Defining Options*.

Creating Tab Controls

You can create a control that allows you to split a form into different tabbed pages. This allows you to use more controls for a single form. You can group your control functions by placing related controls on different tab pages for a single form. You can cut and paste controls from one page onto other pages.

There is a single business view for the form. There is one commit for the form on the OK button. You can use system functions, for example, Set Focus, to add additional functionality for the tab controls.

The following illustration shows a form that has tab pages.

The screenshot shows a software window titled "Customer Service Inquiry". It has a menu bar with "File", "Edit", "Preferences", "Form", "Row", and "Help". Below the menu is a toolbar with icons for "Select", "Find", "Add", "Copy", "Close", "Sequ...", "New...", "Disp...", and "Abo". To the right of the toolbar are "Links", "SO He...", and an "Internet" button. The form has two tabs: "Customer Service Inquiry" (active) and "Additional Selections". The active tab contains several input fields: "Order Number" (with a small grid icon), "Branch/Plant", "Customer PO", "Item Number", "Sold To", and "Ship To". There are also three checkboxes: "Extend Back Order Quantity", "Customer Item", and "As If Currency". Below the input fields is a table with the following headers: "Order Number", "Or Ty", "Order Co", "Line Number", "Hd Cd", "Sold To", "Sold To Name", "Description 1", and "Q". The table body is currently empty.

► To create a tab control

On the form with which you are working

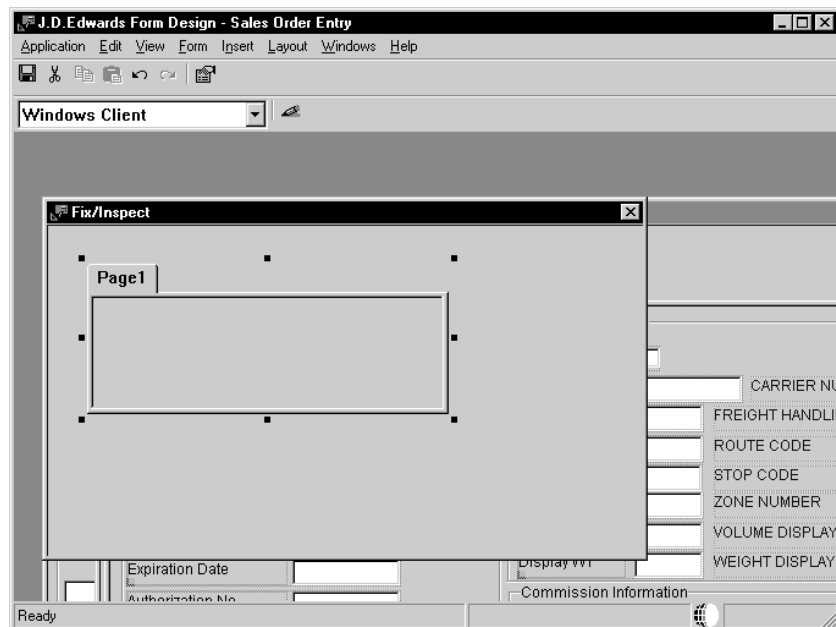
1. From the Insert menu, choose Tab Control.

Page Properties appears to indicate which page of information you are on.

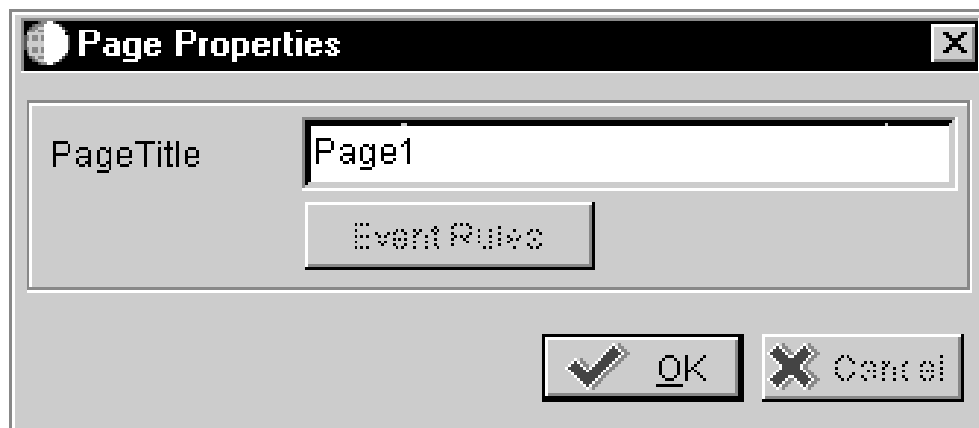


2. On Page Properties, complete the Page Title.

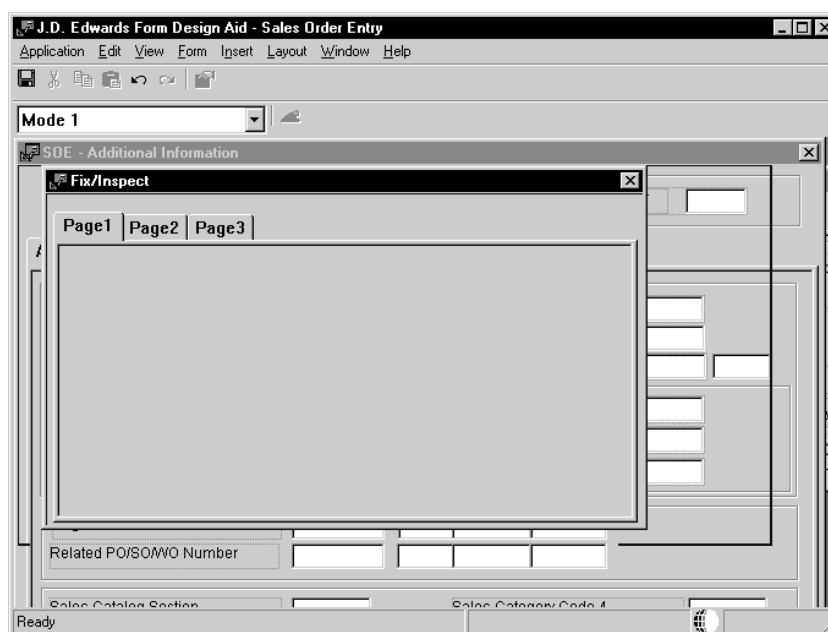
Your form appears with a tab at the top.



3. Position and resize the tab.



4. On Page Properties, change the generic tab name to one that suits your needs.



5. Once you have created the initial tab control, choose Tab Page from the Insert menu to add additional tab pages as necessary.

If you wish to modify your tab pages, click on the tab for the page you want to modify and right-mouse click. You can then insert or delete pages, change the tab page properties, or change event rules specific to that page.

Field	Explanation
Event Rules Title	The default name of the control for use in an event rule. The event rules title defaults to the static text associated with an item. If there is no associated static text and you want to use this control in an event rule, enter a name.
Page Title	Text that appears on the page tab. Avoid abbreviations and acronyms in the page title.

Overriding Data Dictionary Triggers at Design Time

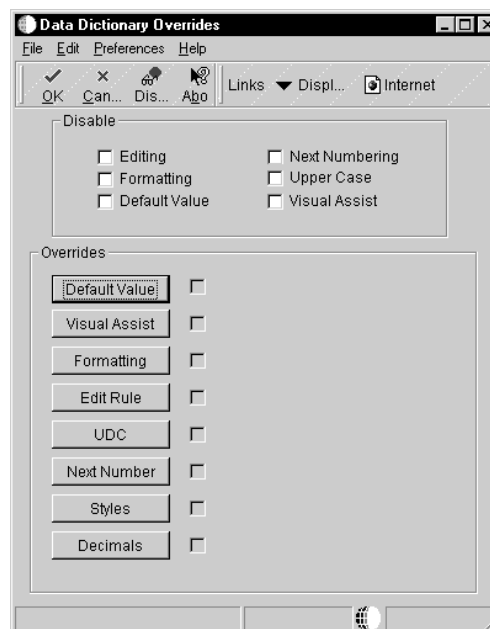
When default triggers are defined in the data dictionary, the application they are attached to automatically executes them at runtime. However, you can override previously defined triggers at the application level, allowing you to further customize how data items behave at runtime.

Use the disable option to turn off data dictionary triggers for a specific item, or to override triggers to change to a new value or option.

► To override previously defined triggers

On the Form, Grid, Edit Control, or UDC Edit Control Properties form

1. Click Overrides.



2. Click any of the following Disable options:

- Editing
 - Formatting
 - Default Value
 - Next Numbering
 - Upper Case
 - Visual Assist
3. Click any of the following to override triggers that were previously defined in the data dictionary:
- CB - Default Value
 - CB - Visual Assist
 - Formatting
 - CB - Edit Rule
 - CB - UDC
 - CB - Next Number
 - CB - Styles
 - CB - Display Decimals

See Also

- *Data Dictionary* for information about setting triggers in the data dictionary

Field	Explanation
Disable Editing	Disables the edit trigger, edit rule, and UDC edits for a field.
Disable Formatting	Disables the format trigger and format rule for a field.
Disable Default Value	Prevents the default value from automatically loading in a field when the user leaves the field blank.
Disable Next Numbering	Disables a next number trigger attached to this data item.
Disable Upper Case	Allows the user to enter lower case characters into a field that has been defined as Upper Case Only in the Data Dictionary.
Disable Visual Assist	Prevents the visual assist button from appearing when the control or grid column is selected.
Override Default Value	Inserts the value into the associated field whenever the user leaves the field blank.

Field	Explanation
Override Visual Assist	Allows the user to control the type of visual button that is associated with a field. If a UDC override is entered this field, it is automatically loaded with the UDC Search Form visual assist. (This override does not apply to the Universal Batch Engine.)
Override Formatting	Allows the user to override the format trigger or format rule that is associated with a field/control.
Override Edit Rule	Allows the user to override the edit trigger and edit rules associated with this field. If edit rules or procedures are entered for control, then the UDC override is disabled. A field can have either an edit trigger, edit rule, or UDC edit; it cannot have more than one of these types of edits.
Override UDC	Allows the user to specify the UDC table to which a field is validated against. When a UDC override is entered, the override search form is automatically assigned to the UDC Search Form. Entering an override disables the edit rule button and clears any edit rules or procedures.
Override Next Number	Allows the user to specify the location and starting value for next numbering on a field.
Override Styles	Allows the user to override the styles, Allow Blank Entry and Upper Case Only. Allow Blank Entry is used in conjunction with a UDC edit. If the UDC table for which a field is validated does not contain a blank value, then specifying the allow blank entry flag allows the UDC field to contain a blank value without generating errors. The upper case only flag causes a field to convert all entry values to upper case.
Override Decimals	Allows the user to control the number of decimals instead of the traditional two decimal places. This override pertains to display decimals only.

Overriding a Default Value Trigger

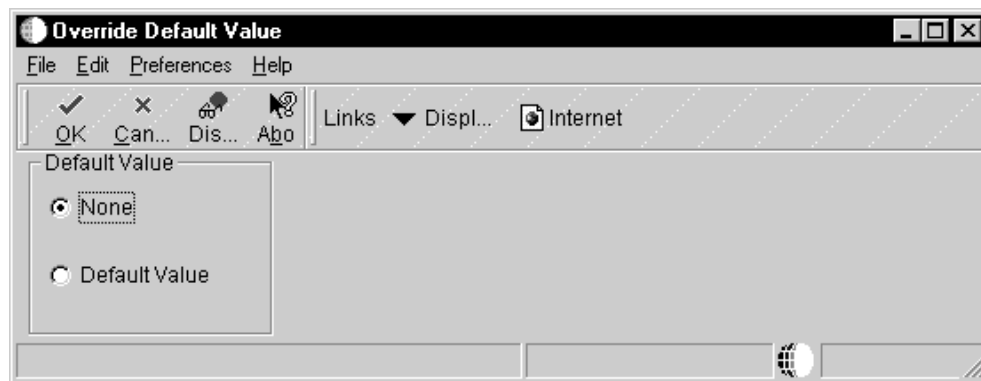
You can override a default value trigger to assign a different default value for a field or to allow no default value.



To override a default value trigger

On Data Dictionary Overrides

1. Click the Default Value button.
2. On Override Default Value, click one of the following:
 - None
 - Default Value



Field	Explanation
Default Value	Indicates whether there is a default value on the field. Choose either of the following options: • None - no default value is assigned • Default Value - assigns a default value to the field. Specify the default value that this field will have.

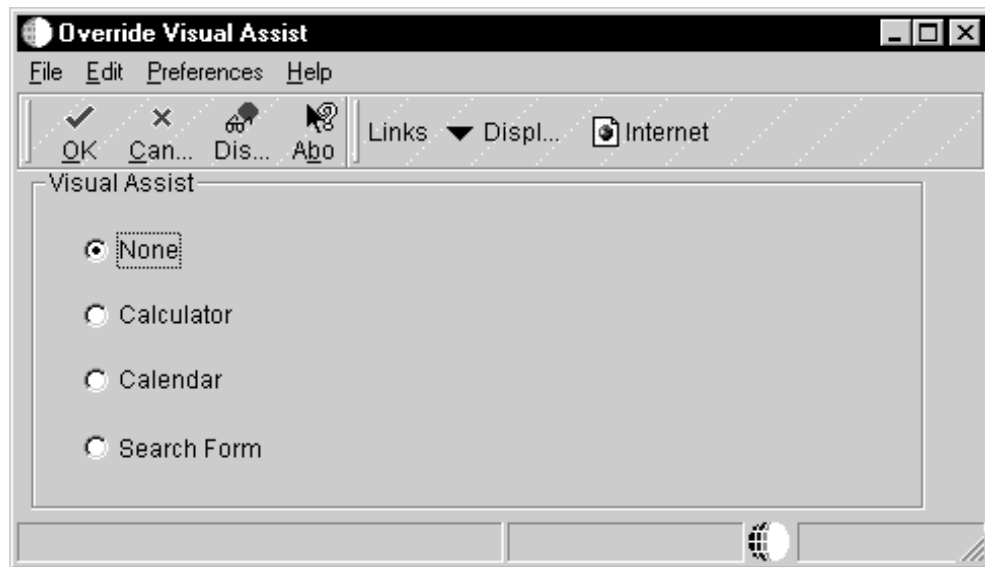
Overriding a Visual Assist Trigger

You can override a visual assist trigger.

► To override a visual assist trigger

On Data Dictionary Overrides

1. Click the Visual Assist button.
2. On Override Visual Assist, click one of the following:
 - None
 - Calculator
 - Calendar
 - Search Form



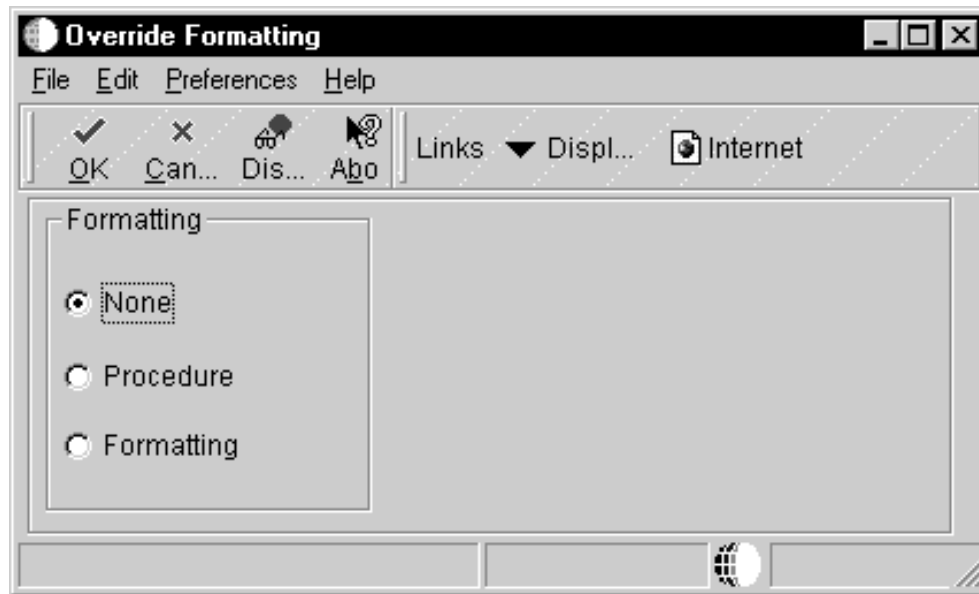
Field	Explanation
Visual Assist	Indicates whether there is a visual assist on a field and the type. Choose from the following options: • None - no Visual Assist is assigned • Calculator - assigns a calculator to the field. Use this on all numeric data items where a user might need to type in an amount. • Calendar - assigns a calendar to the field. Use this on data items where a user might need to type in a date. • Search form - assigns a search form to the field. Use this on all fields where the valid values for the field are found in a file.

Overriding a Formatting Trigger

► To override a formatting trigger

On Data Dictionary Overrides

1. Click the Formatting button.
2. On Override Formatting, click one of the following:
 - None
 - Procedure
 - Formatting



Field	Explanation
Display Rule	Indicate whether or not there is a Formatting rule on the field. Choose from the following options: • None - no Formatting rule • Procedure - assigns a business function ID to the field. Specify the business function procedure in the field that becomes available when you choose this option. Use the visual assist to view and select a business function. • Formatting - assigns a display rule to the field. Specify the display rule in the fields that become available when you choose this action. Use the visual assist to view and select rules.

Overriding an Edit Rule Trigger

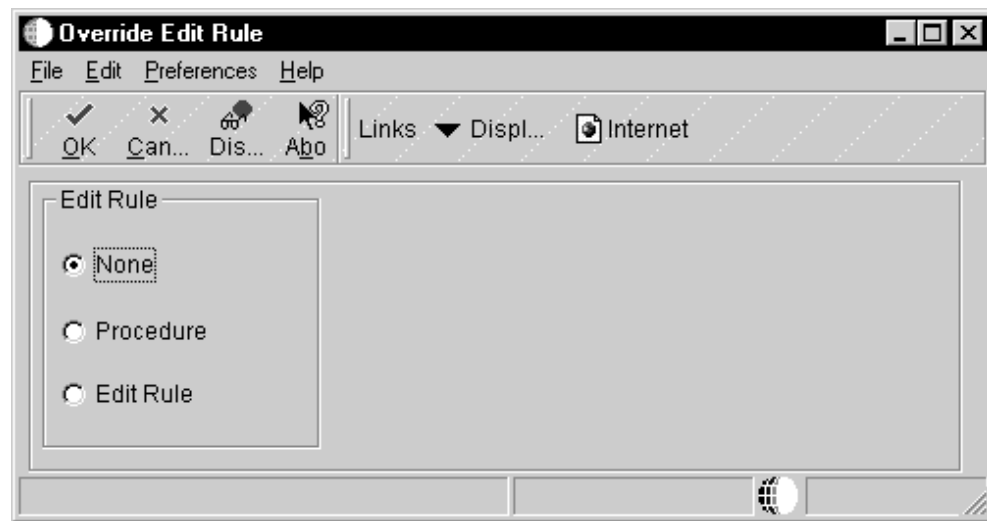
You can override an edit rule trigger.

► To override an edit rule trigger

On Data Dictionary Overrides

1. Click the Edit Rule button.
2. On Override Edit Rule, click one of the following:
 - None
 - Procedure

- Edit Rule



Field	Explanation
Edit Rule	Indicate whether there is an Edit Rule on the field and the type. Choose from the following options: • None - no Edit Rule is assigned • Procedure - assigns an Edit Rule based on a business function. Use Business functions when you need special processing that cannot be done through one of the five display rules. For example, if you would like to format the appearance of the account number, attach a business function called Display Account Number. Specify the business function Procedure in the field that becomes available when you choose this option. Use the visual assist to view and select a business function. • Edit Rule - assigns an Edit Rule. Specify the edit rules in the fields that become available when you choose this option. Use the visual assist to view and select rules.

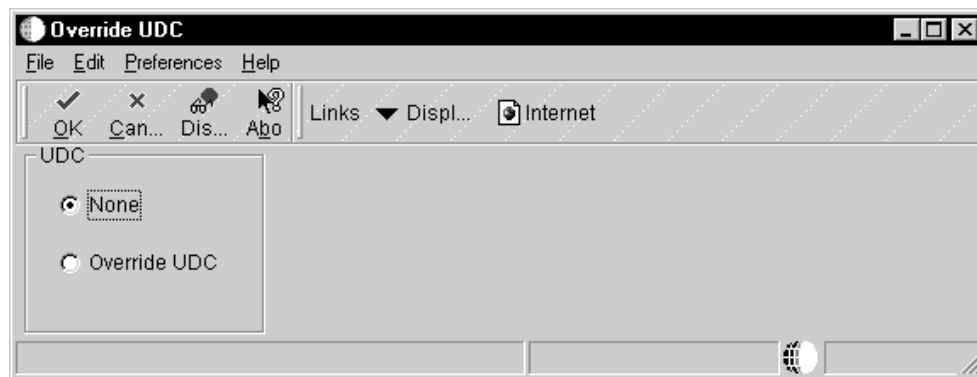
Overriding a User Defined Codes Trigger

You can override a user defined codes trigger.

To override a User Defined Codes trigger

On Data Dictionary Overrides

1. Click the UDC button.
2. On Override UDC, click one of the following:
 - None
 - Override UDC



Field	Explanation
UDC	Indicate whether there is a User Defined Code override on the field. Choose either of the following options: • None - no UDC table is available • Override UDC - assigns a user defined code override to the field. Specify the System Code and Code Type for the UDC table.

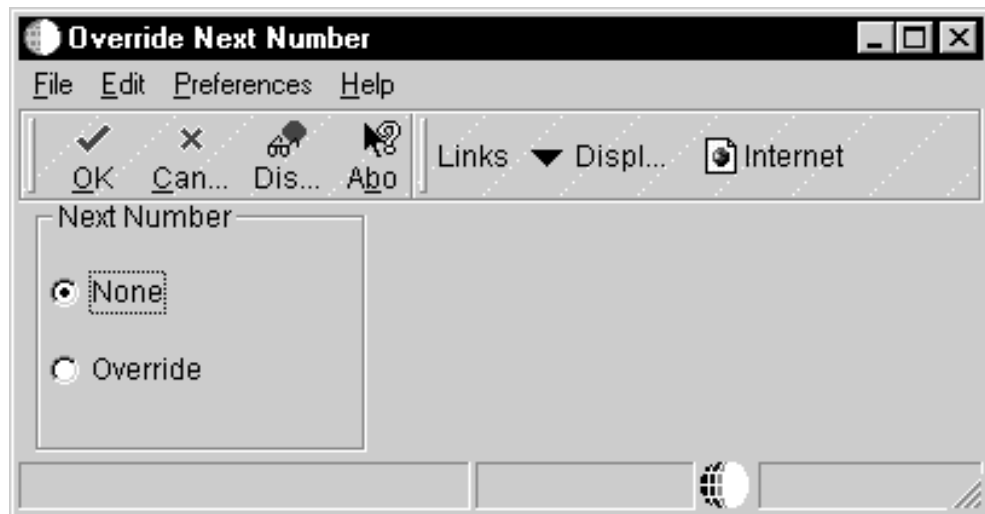
Overriding a Next Number Trigger

You can override a next number trigger.

To override a next number trigger

On Data Dictionary Overrides

1. Click the Next Number button.
2. On Override Next Number, click one of the following:
 - None
 - Override



Field	Explanation
Next Number	Indicate whether there is a Next Number override on the field. Choose either of the following options: • None - no Next Number is assigned • Override - assigns next number logic to the data item. Specify the Next Number System Code and Next Number Index the data will use in the Next Numbers file.

Overriding a Styles Trigger

You can override a styles trigger to change the way information on a form displays.

► To override a styles trigger

On Data Dictionary Overrides

1. Click the Styles button.
2. On Override Styles, click one of the following:
 - Allow Blank Entry (Y/N)
 - Upper Case Only (Y/N)



Field	Explanation
Allow Blank Entry	Turning this option on allows a blank value to be written to the database under the following conditions: (1) If the field is edited against a UDC table, a blank value will be allowed regardless of whether a blank value is valid for the table. (2) If the field is specified to be a mandatory entry, a blank value will be allowed as a valid entry.
Upper Case Only	If the value of this field is a Y, the user will be allowed to enter only upper case characters in the entry control.

Overriding a Decimal Trigger

You can override a decimal trigger to change how decimals display.

► To override a decimal trigger

On Data Dictionary Overrides

1. Click the Decimals button.
2. On Override Display complete the following field:
 - Display Decimals

Field	Explanation
Display Decimals	Designates the number of decimals in the currency, amount, or quantity fields the system displays. For example, U.S. Dollars would be 2 decimals, Japanese Yen would be no decimals, and Cameroon Francs would be 3 decimals.

Designing Forms Using Multiple Modes

You can use control modes to develop an application with multiple interfaces. This reduces the need to maintain several different versions of the same application. You can create one base application and use modes to modify the application for different interfaces. You can enable or hide controls on forms for each mode. Only visibility properties for controls and columns are different for different modes. If you show hidden fields, they appear only for the mode you are in. All other properties are the same and are common for all modes. All fields are enabled and shown on all forms.

Control modes are not recognized by the Windows runtime engine, only the Java runtime. Mode 1 is the default mode. You attach an application to a menu to run. This menu allows you to run an application in different modes. When you run an application over the Web, the application runs in mode 1 by default and another mode if one is specified. If you attach an application to a Windows menu, the Windows runtime engine ignores any modes that are specified and runs the application in mode 1. Try to use modes consistently throughout your applications.

Viewing Forms in a Particular Mode

You can view the forms you are developing in one of three modes:

- Mode 1
- Mode 2
- Mode 3



To view forms in a particular mode

On Forms Design

1. From the View menu, choose Control Modes
2. From the mode drop down menu that appears, choose one of the following mode options:
 - Mode 1
 - Mode 2
 - Mode 3

The forms and controls particular to the mode you have chosen appear.

You usually develop your applications in Mode 1 and customize them as needed for Mode 2 and Mode 3.

Setting Control Mode Properties

You can set mode properties individually for each control on a form. The default display for all modes is enabled. This allows you to enable or hide controls for each of the following modes:

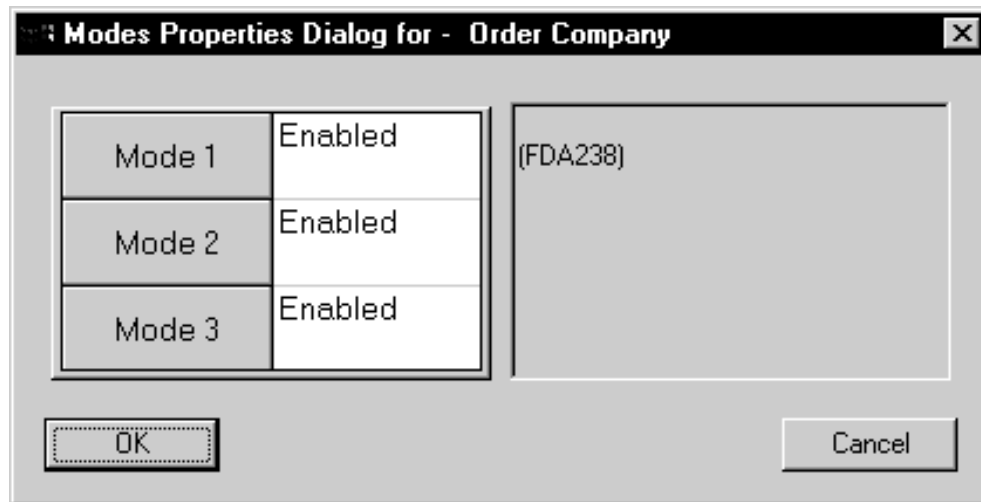
If you hide a control, it is disabled. This means that the event rules and logic for that control will not run, so be very careful when you hide controls.

- Mode 1
- Mode 2
- Mode 3



To set control mode properties

1. On the form with which you are working, right-click on the control you wish to enable or hide.
2. Choose Modes from the menu that appears.



3. On Modes Properties double-click on the Enabled box beside one of the following modes:
 - Mode 1

- Mode 2
 - Mode 3
4. From the menu that appears, choose Enabled or Hidden.

Controls are automatically enabled unless you change them to hidden.

Using Text Variables

Text variables are stored as strings and can be used as an alternative to hard coding text strings in assignments. Because text variables are not hard-coded, they are easier to maintain. Following are some of the ways you can use text variables:

- To reuse forms, for example, you can reuse message forms and display the appropriate message depending on specific conditions.
- To reuse grid columns instead of hiding and showing them by changing the column heading text.

To use a text variable you:

- Add the text variable itself
- Use system functions to attach the text variable or use the assignment to assign it

Adding a Text Variable

You can use the Text Variable form to add all of the text variables you want to use for a specific form.



To add a text variable

In Form Design, on the form for which you wish to add a text variable

1. From the Form menu, choose Text Variables

Current text variables associated with the selected form appear.

Text String	
On Call	
Tech	

2. Under Text String, choose the last empty row and enter the text string you wish to use.
3. Tab to the next row to create another text variable.

If you try to create a text string that is named the same as another variable on the same form (or on the same section in Report Design) an error occurs. Rename the variable to clear the error.

4. Click Save to save your variables.

You can modify and delete text variables. To modify a text variable, simply type over it and save the new text.

If you delete a text variable that is referenced in event rules be sure to delete its reference in event rules.

Attaching or Assigning a Text Variable

You can use system functions to attach text variables. You can use a system function to change the text for a control, to change the name on the form, or to change the text in a grid column heading. The following system functions are typically used for text variables:

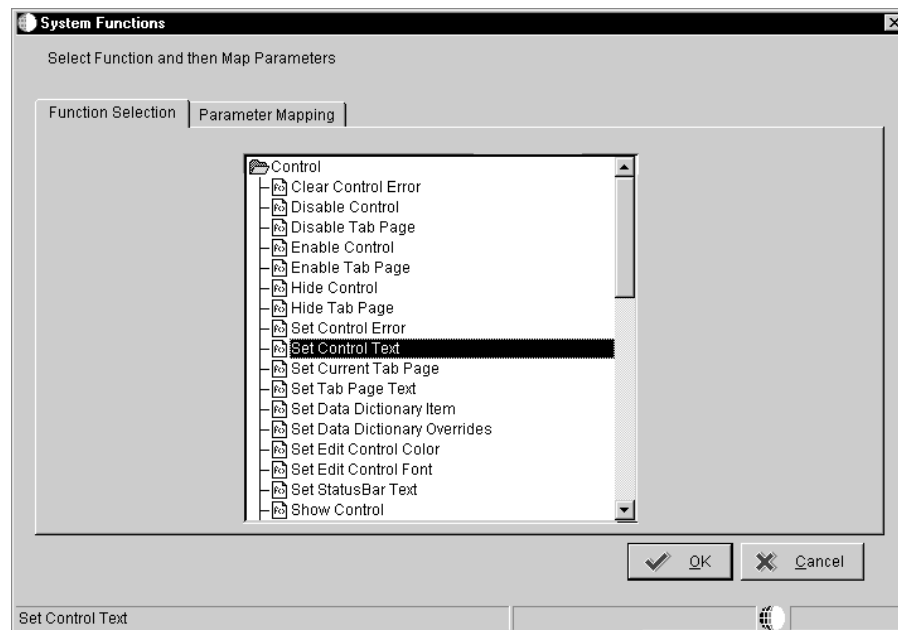
- SetControlText
- SetGridColumnHeading
- SetFormTitle

The available text variables appear in the Available Object list for System Functions. Available text variables also appear when you use assignments.

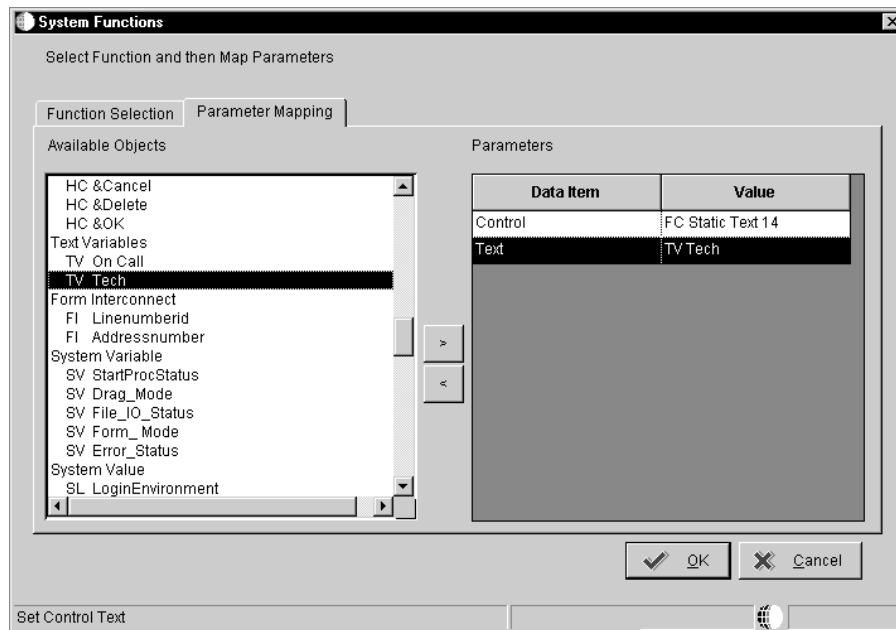
► To attach a text variable

If you want to attach text to a control, on the form you are working with

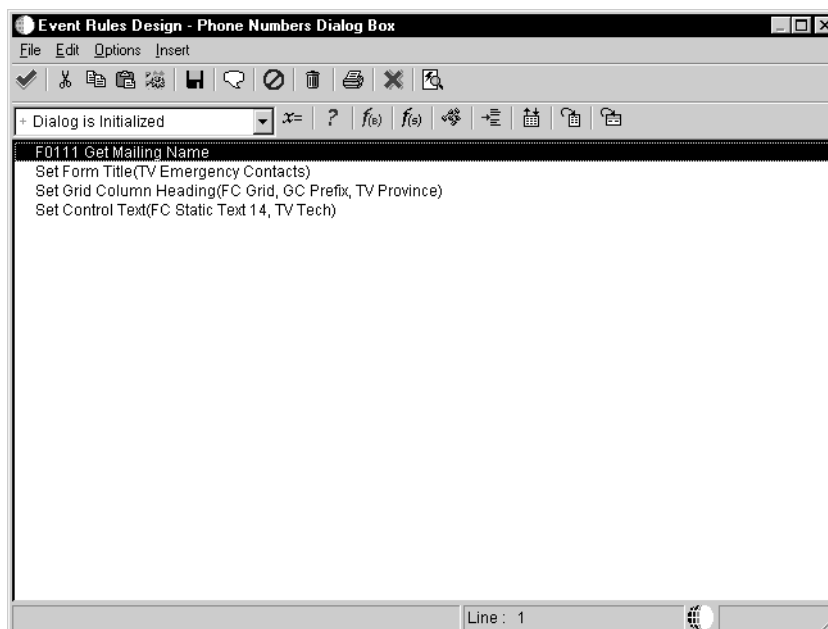
1. From the Edit menu, choose event rules
2. From the events list, choose the event to which you want to attach the text variable.
3. Click on the System Function button.



4. Choose the Set Control Text system function.



5. Complete the parameters.



Refer to *Attaching Functions* in this guide for more information about attaching system functions.

You can also assign a text variable to the edit fields of data dictionary and business view items. Refer to *Creating Assignments* in this guide.

Using Quick Form

Use Quick Form to easily place multiple database fields on a form, rather than choosing each data item individually. Simply choose one or more data items and Quick Form automatically places each field on the new form simultaneously.

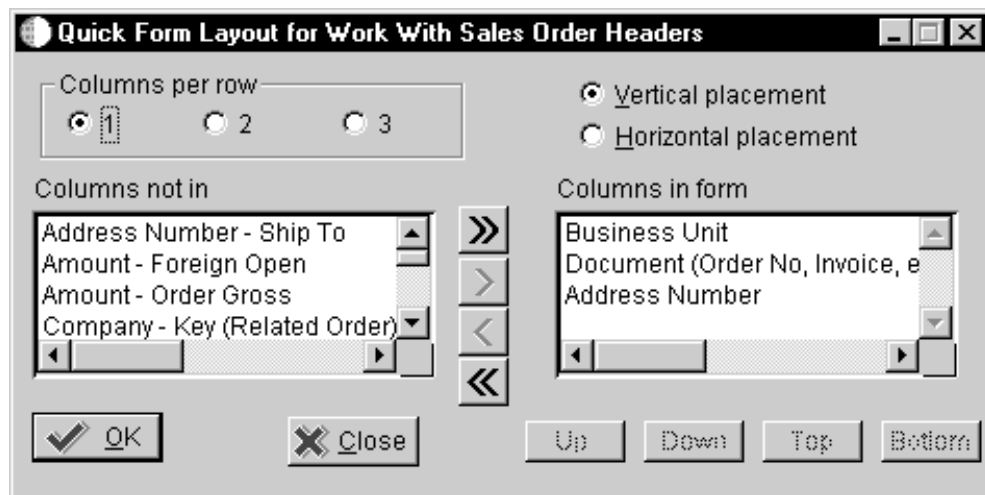
Depending on the number of selected fields, you might need to resize the form or move and align fields to achieve the desired layout.

► To use Quick Form

On Forms Design

1. From the Form menu, choose Quick Form.

The title bar reflects the business view associated with the form.



2. Click the following options:
 - Columns Per Row
 - Vertical/Horizontal Placement
3. Choose and sort columns from the business view.

4. Click OK to place the fields on the form.

Quick Form remains open so you can adjust the arrangement by changing the columns per row and the vertical and horizontal placement.

5. Click Close when you are finished.

Field	Explanation
Columns Per Row	Specifies how many columns will be used to arrange the fields.
Vertical/Horizontal Placement	Specifies the space between the horizontal and vertical grid coordinates. The default spacing is 8 pixels between each horizontal and vertical gridline.
Columns not in form	Indicates the data items in the business view that are associated, but not yet placed on the form. Use none, some, or all the items in the business view. For example, on Find/Browse forms you can display fields only in the grid control and none on the form.
Columns in form	Lists the selected data items from the business view that are placed on the form.

Processing Media Objects

Media Objects are used to link information or “attachments” to application transactions.

You can use standard processing for media objects to bypass all event rules that are required to implement media objects. All of the required information is gathered for a form in Forms Design Aid and does not require the entry of any event rules. Standard processing does the following:

- Allows implementation of media objects at the form level with no event rule coding required
- Standardizes the usage of media objects across forms
- For any grid, places a paper clip icon on the row header if a media object is defined for that row
- For a form, places an icon in the status bar if a media object is defined for the form
- Allows you to attach documents to the form or to a row in the grid
- Allows you to double click on the paper clip in a row to start media objects for that row
- Allows you to click on the paper clip in the status bar to start media objects for the form

If you choose not to use standard processing for a form, you can still use event rules and system functions to make media objects work.

The F00165 table is used to store link records for media objects and imaging. You must define your media object data structure using a unique key structure so that the F00165 table can store data correctly. The layout of the F00165 table is as follows:

GTxxx || F4211Keys || The media object text

GTxxx is the naming convention when defining a media object data structure. The F4211Keys portion is what the system uses to access the unique media object attachment for that particular record. The keys here typically match what the unique key would be in the F4211 for each detail line. The media object text portion is the actual text attachment that would store information typed in by the user.

Imaging

Imaging allows you to use images from third-party software.

Imaging is done on a form-by-form basis for performance reasons. When you attach an image to a document, the media object form includes the vendor's software as a choice.

When you use a third-party vendor, the F00165 file stores the reference to image attachments, but the third-party software controls the search and retrieval of images.

Before You Begin

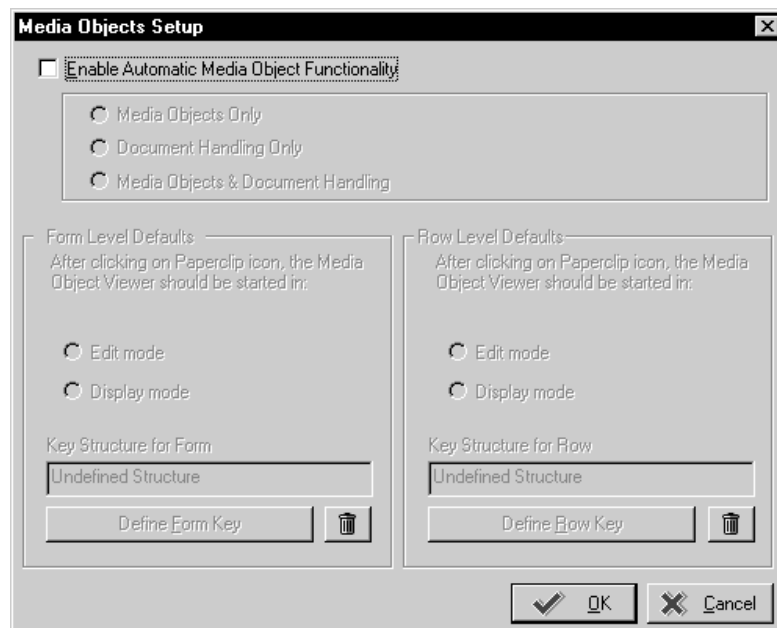
In order to see the media object paper clip column on your form, you must turn off the Hide Row Numbers option in the Grid properties for the form in Form Design.

Using Standard Processing for Media Objects

► To use standard processing for media objects

On Form Design

1. From the form menu, choose Media Objects Setup.



2. Choose Enable Automatic Media Object Functionality.

This enables imaging and opens the other fields on the form

Media Objects Setup

☒ Enable Automatic Media Object Functionality

☐ Media Objects Only
☐ Document Handling Only
☒ Media Objects & Document Handling

Form Level Defaults
 After clicking on Paperclip icon, the Media Object Viewer should be started in:

☒ Edit mode
☐ Display mode

Key Structure for Form
 Undefined Structure
 Define Form Key [trash icon]

Row Level Defaults
 After clicking on Paperclip icon, the Media Object Viewer should be started in:

☒ Edit mode
☐ Display mode

Key Structure for Row
 Undefined Structure
 Define Row Key [trash icon]

[OK] [Cancel]

3. On Media Objects Setup, click one of the following:

- Media Objects Only
- Document Handling Only

Use the Document Handling Only option if you are developing a form that is enabled for media objects via functionality in event rules and you wish to bypass standard processing. If you wish to enable standard processing later, you must delete all of the event rules for media objects and click the Media Objects & Document Handling option.

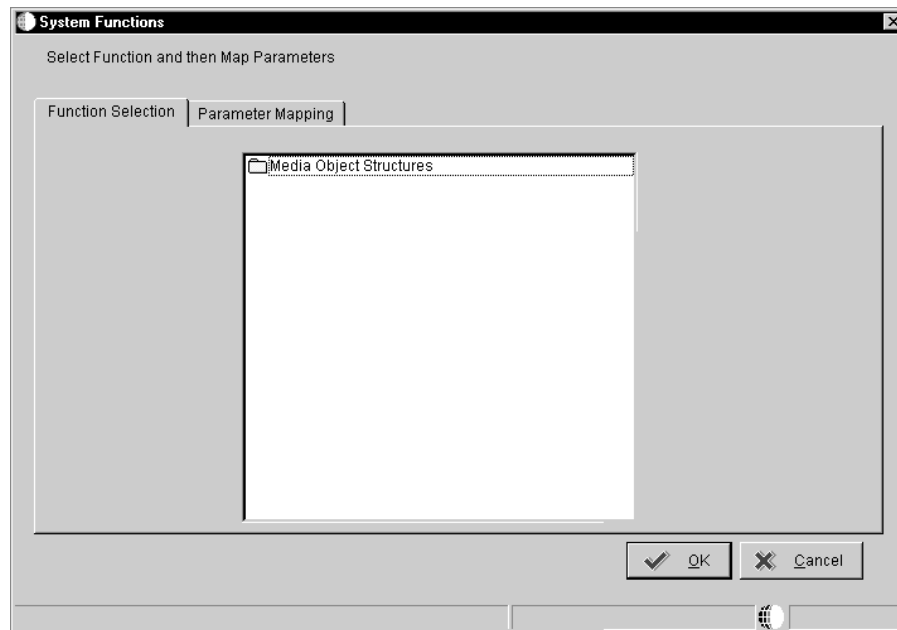
- Media Objects & Document Handling

4. Click Edit mode or Display mode.

Edit mode allows the user to make changes, while display mode is read-only.

5. Click the Define Form Key button.

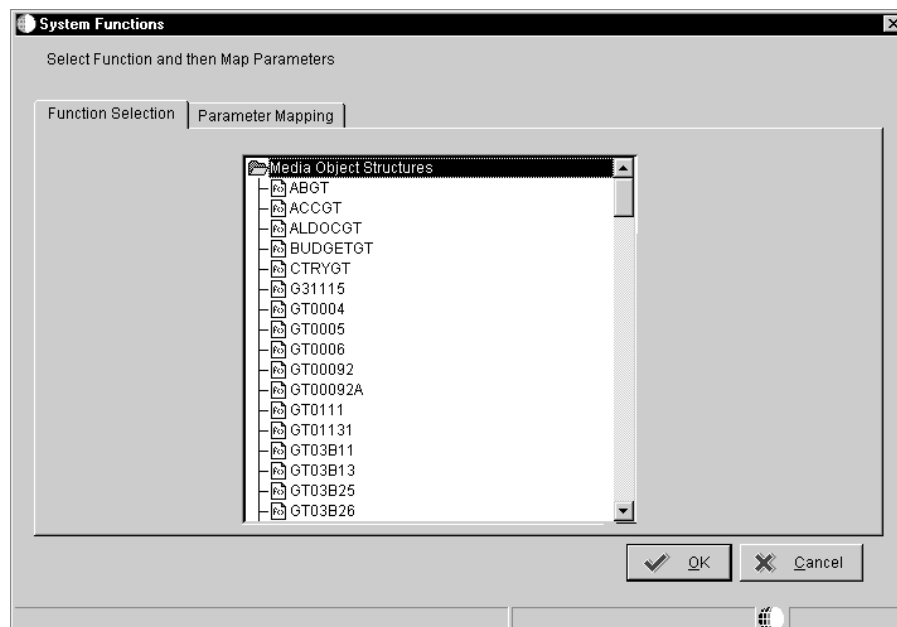
The System Functions form appears. This form is identical to the parameter definition form used to define system functions in Event Rules, except only the Media Objects header is displayed.



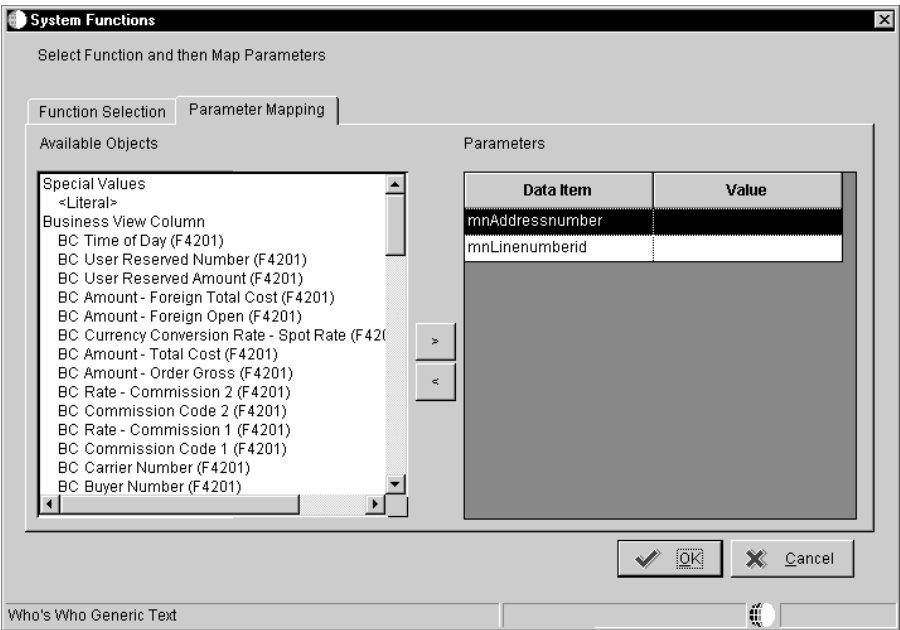
The key structure is used to store link records in the F00165 file. These links are critical to proper functioning of Imaging.

6. Double-click on the Media Object Structures folder.

A list of all of the currently defined data structures for Media Objects displays.



7. Choose the appropriate structure and define it.



Field	Explanation
Enable automatic media object function	Check this to define the media object. When this is checked, the form level defaults and row level defaults are enabled so that you can define the media object. When this is not checked, all fields on this form are dimmed.
Media Objects Only	After enabling automatic media object functionality, choose one of the following options: - Media objects only - Document handling only - Media object and document handling Use option, Document Handling Only, if the form uses event rule logic rather than standard processing for media objects. If you want to enable standard processing later, you must delete all event rules for media objects and then select Media Objects and Document Handling.
Edit Mode	Allows the user to change the media object.
Display Mode	Changes for the media object are not allowed. The user can only view the media object.

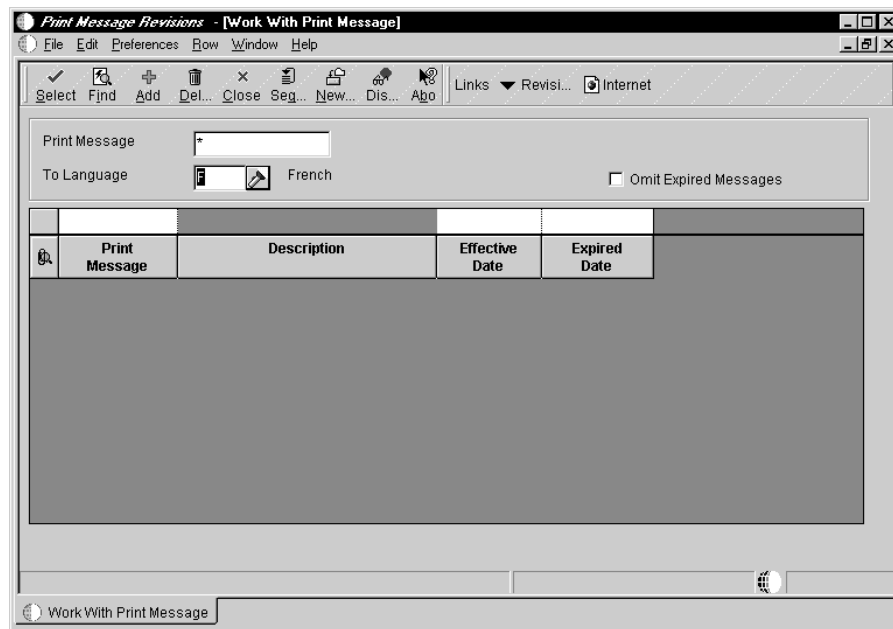
Language Considerations for Media Objects

You can add media object (attachments) for a particular language. Most J.D. Edwards applications are media object enabled for languages.

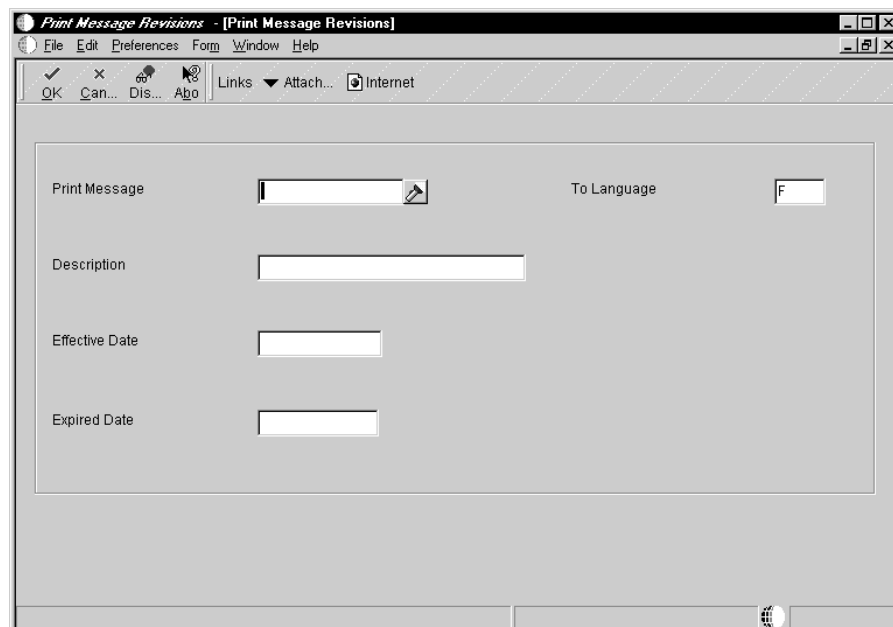
► To add a language specific media object attachment

On the application which you wish to use

1. Type a language in the filter field.



2. Click Add.



3. Add multiple records if you want the attachment for multiple languages or base.

If you create a custom application that you want enabled for media object language handling, you must include a data item language preference (alias LNGP) in the generic text data structure you create.

Example: Language Considerations for Media Objects

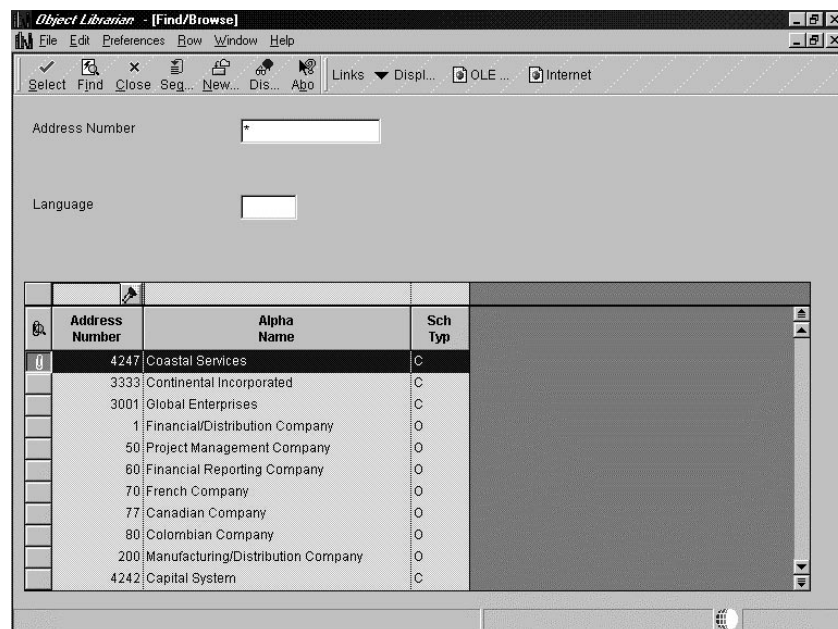
When you design an application, you can allow the end user to add separate and unique media objects, to the same record or different records, based on the language chosen.

If language (LNGP) *is not* a database column, then you define your media object (GT) data structure to include language as part of the data structure. You place a data dictionary control (LNGP) on the application as a filter field which should then be loaded with the system value for language. When you design your application this way, you attach two separate media objects, based on the language, to the same record.

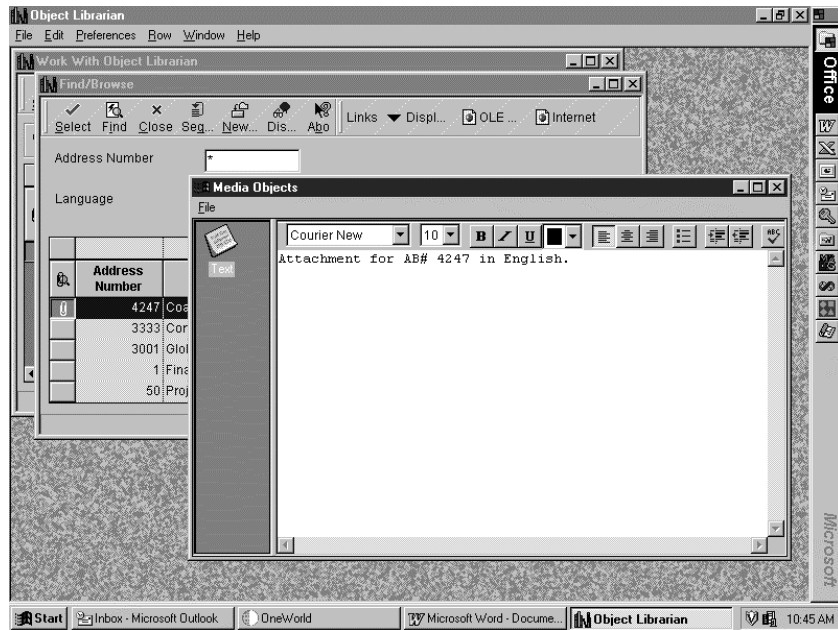
If language (LNGP) *is* a database column, then you include LNGP (database) as a filter field, but you must add separate record to the database table along with its media object attachment. The media object data structure still contains language as part of the key to retrieve the media object attachment. In both cases, the language filter fields (LNGP) needs to be loaded with the system value for language.

In both of the above situations, the language filter fields (LNGP) must be loaded with the system value for language.

For example, the following Address Book application includes Address Number and Language as filter fields. Note that the language control *is not* from the database, but is a Dictionary control.

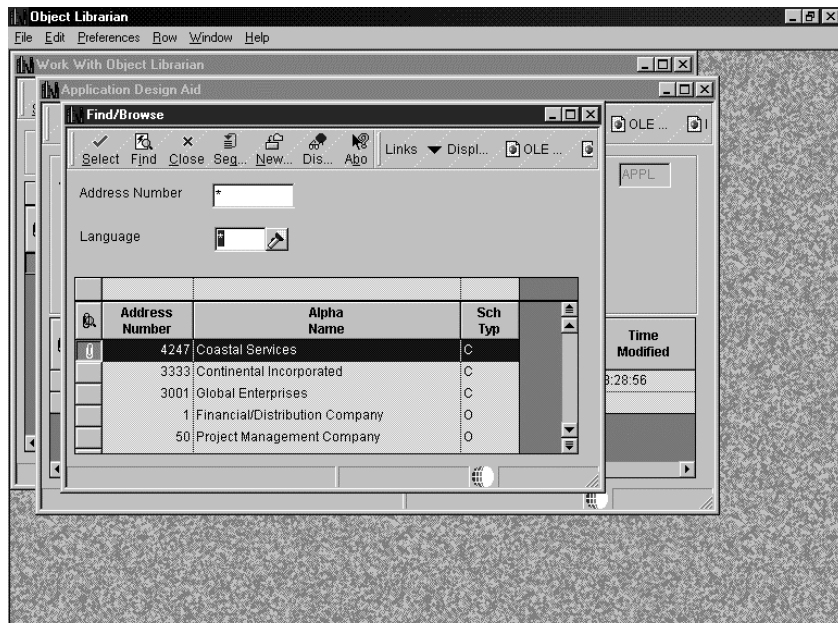


For a blank language (English), Address Book record number 4247 has a media object attachment in English.

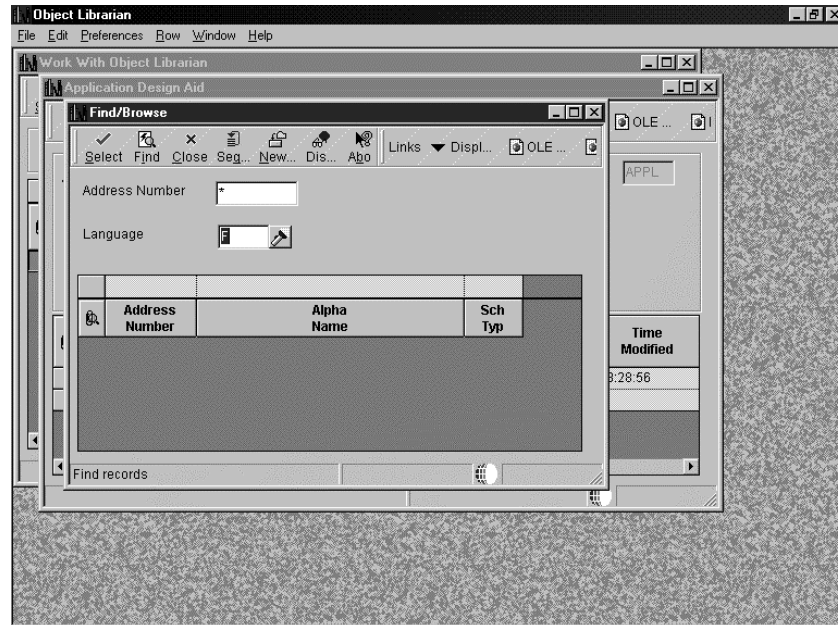


If the same record was accessed by a user whose language preference was set to French (F), the filter field would get loaded with “F” and the user would see a totally different Media Object attachment for the same record. The Address Book would not have two separate records based on the language, but only one record in English. For the French Media Object attachment a separate media object attachment record is kept for the same Address Book record in the F00165 table. It can be accessed using the language filter control field.

In this example, the ABLNGP field/column from the Address Book table is used as the filter field on the form. The Language field on the form *is* from the database and not a Data Dictionary item.



When you search for records using blank or “*” (English or “ALL”) as the base language, you see all records in the Address Book with their corresponding media object attachments. You use the same media object data structure as you did with the previous case, which contains the Address Number and Language data items. You still load the user’s language preference in the language filter field, but now when the application fetches records from the database it is using the language as part of the key to select records from the Address Book. In this example, if there are no records in the Address Book with the language preference for French “F” you will not see any records and you then must add records using French as the language.



Because the Address Book is only keyed using Address Number and not the language we cannot add a record for Address Book# 4247 in French, but we can add new records with new Address Book Records with new AB #'s with the language Preference of French and then be able to see only those records.

For any database table containing language as part of its key, you can attach media objects functionality for records with different languages. For example, one record for English and a copy of the record for French with unique media object attachments. For tables that do not include language as part of the key to that table, you can have media object languages, for example item master print messages GH4112, P40162.

Testing a Form

You should test a form to verify placement of controls on the form. When you test a form, it displays exactly as it will appear in the application.

Test mode is only for viewing. You cannot modify a form in test mode. Test works for the default Windows mode.

► To test a form

On Forms Design

1. From the Form menu, choose Test.

The form is displayed as it will appear in the working application.

A screenshot of a software window titled "Sales Detail Line Defaults". The window has a menu bar with "File", "Edit", "Preferences", and "Help". Below the menu bar is a toolbar with icons for "OK" (checkmark), "Can..." (X), "Dis..." (globe), "App" (mouse cursor), "Links" (dropdown arrow), "Displ..." (dropdown arrow), and "Inte" (globe). The main area of the form contains two sections. The first section has two labels, "Detail Branch/Plant" and "Customer PO", each followed by a text input field. The second section is titled "Dates" and contains four labels: "Requested", "Scheduled Pick", "Promised Delivery", and "Cancel Date", each followed by a date input field. At the bottom of the window is a status bar with a globe icon.

2. To return to design mode, click Close or Cancel in the test form.

Language Preview

You can preview forms in foreign languages. You must have the base language installed on your machine to do this.

To preview a form in another language

On the form you want to preview

1. From the Form menu, choose Language preview.
2. Choose the language in which to display the form and click OK.
3. To continue designing the form, close the language preview form.

Event Rules

Event Rules Design

Use Event Rules Design to create business logic for an application. For example, create event rules that:

- 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 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, data validations, and record retrieval

This section describes the following:

- ☐ Runtime processing
- ☐ Working with Event Rules Design
- ☐ Working with If and While Statements
- ☐ Working with event rule variables
- ☐ Attaching Functions
- ☐ Creating Form Interconnections
- ☐ Creating Report Interconnections
- ☐ Creating Assignments
- ☐ Table I/O
- ☐ Table Event Rules
- ☐ Working with asynchronous processing
- ☐ BrowsER



Before You Begin

Before working with Event Rules Design, you must:

- ☐ 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
- ☐ Think about the purpose of each form used in the application
- ☐ Answer the following four questions:
 - What logic is required?
 - For which control are you creating logic?
 - For which event will the logic occur?
 - Which runtime structures are affected?

Understanding Controls

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

Understanding Events

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

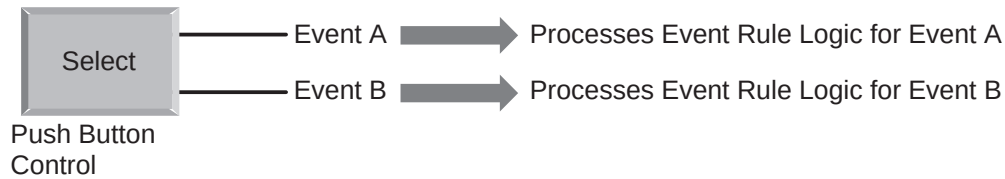
Understanding Form Processing

Form processing refers to the business logic associated with each form. By default, each type of OneWorld form automatically includes basic processing for various events. Then, using Event Rules Design, you can build additional logic. Form processing depends on the occurrence of specific events, such as initializing a form or changing the value of a field.

See Appendix B: Form Processing

Understanding Event Rules

Event rules are logic statements that you can create and attach to events. They are initiated when events occur at runtime. The following illustration shows this relationship.



You can attach multiple event rules to one event.

Types of event rules include:

- If/Else/End if conditional statements
- While loops
- Assignments to statements
- Calls to encapsulated functions
- Form or report interconnections
- Calls to system functions
- Table I/O operations

OneWorld uses two different types of event rules: business function event rules and embedded event rules. Each is described below.

Business Function Event Rules

Business function event rules are encapsulated, reusable business logic created using event rules, rather than C programming. They are stored as objects and are compiled. Business function event rules are sometimes called Named Event Rules (NER).

Embedded Event Rules

Embedded event rules are specific to a particular table, interactive application, or batch application. They are not reusable. Examples include form-to-form calls, hiding a field based on a processing options value, 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 are attached to a table through Table Design Event Rules. The logic attached to a table is executed whenever any application initiates that database event. For example, you might have rules on a master table to delete all children when a parent is deleted, which maintains referential integrity. Any application that initiates a delete of that table will not need the parent/child logic imbedded in the application because it resides at the table level.

Refer to the online help Published API section for information about individual events.

Understanding the 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 J.D. Edwards 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 only valid within the bounds of If and Endif
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

Different form types have specific event rules that are typically used for that form type.

Find/Browse

Form-Level Events

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Write Grid Line - Before
- Last Grid Record Has Been Read
- Delete DB Record Verify - Before
- Delete DB Record Verify - After
- Delete Record from DB - Before
- Delete Record from DB - After
- End Dialog
- Write Grid Line After

Grid-Level Events

- Delete Grid Rec Verify - Before
- Delete Grid Rec Verify - After
- Delete Grid Rec from DB - Before
- Delete Grid Rec from DB - After
- All Grid Recs Deleted from DB
- Double Click on Row Header
- Set Focus on Grid
- Kill Focus on Grid
- Row is Exited

Fix/Inspect

- Dialog is Initialized

- Clear Screen Before Add
- Post Dialog is Initialized
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Delete DB Record Verify - Before
- Delete DB Record Verify - After
- Delete Record from DB - Before
- Delete Record from DB - After
- End Dialog
- Clear Screen After Add

Header Detail and Headerless Detail

Form-Level Events

- Dialog is Initialized
- Clear Screen Before Add
- Post Dialog is Initialized
- Grid Record is Fetched
- Add Row
- Write Grid Line - Before
- Last Grid Record Has Been Read
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Delete DB Record Verify - Before
- Delete DB Record Verify - After
- Delete Record from DB - Before
- Delete Record from DB - After
- End Dialog
- Write Grid Line After
- Clear Screen After Add
- Row is Exited and Changed - Asynch

Grid-Level Events

- Add Grid Record to DB - Before
- Add Grid Record to DB - After
- All Grid Recs Added to DB
- Update Record to DB - Before
- Update Record to DB - After
- All Grid Recs Updated to DB
- Delete Grid Rec Verify - Before
- Delete Grid Rec Verify - After
- Delete Grid Rec from DB - Before
- Delete Grid Rec from DB - After
- All Grid Recs Deleted from DB
- Add Last Entry Row to Grid
- Double Click on Row Header
- Set Focus on Grid
- Kill Focus on Grid
- Row is Exited and Changed - Inline
- Row is Exited

Runtime Processing

Runtime processing refers to how events (such as initializing a form, clicking a button, and tabbing into or out of a field) and their attached event rule logic are evaluated at runtime. Event rule logic is attached to events, which in turn are attached to controls.

Forms Design provides several different form types, each including predefined fields and functionality specific to the form type. For example, the Find/Browse form automatically includes a Find menu option or toolbar button with appropriate functionality attached. When you type search text into a filter field or Query by Example (QBE) field and you click the Find button on the toolbar, the runtime engine processes logic to fetch a record .

You should understand the different field types and associated functionality that characterize each form type. This prevents you from unnecessarily generating event rules. Should you want to customize existing functionality, use this section as a reference to assist you in your modifications.

Runtime Structures

Runtime structures are structures or blocks of memory that hold data in memory as the data is read, processed and written to the database. You should know what is happening behind the scenes on each form. You should know what is in a runtime structure at a given event point.

For example, if a form contains hidden controls, memory is allocated for those controls even though they are not visible on the form. When you pass processing option values into a form, memory is allocated to store the processing option value so it can be passed to the form.

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 comprise the runtime data structure for a form.

During runtime processing, data is stored 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 until it is no longer needed. Then data can be cleared to process another record.

The following available objects are defined:

BC	A column in the business view. Business view columns for both the form view and the grid view will appear in this list. These columns are filled with values from the database when a fetch is performed and are the values saved to the database on an add or update.
GC	A column in the grid. The row that the value is referencing will depend on which event is accessing the GC. During the fetch cycle it will be the row being fetched. It is usually the selected row. In some circumstances, these objects are also used to denote a particular physical column in the grid instead of a value, as with the Set Grid Line Font system function.
GB	The grid buffer is one row of data space independent of the lines being read from the database and written into the grid. It allows you to manipulate column data for a line you wish to insert or update without affecting the grid's present state. This space is accessed through the available object Grid Buffer Column (GB), which appears after the Grid Columns (GC) in the list of available objects in event rules. There is only one instance of each GB column per grid. It is not associated with any particular line.
FC	A control on the form. If the control is a database item, this field will correspond to a BC object. Furthermore, if the control is not a filter, the FC object will represent the same value as the BC object, and changing one will result in modifying both.
FI	A value passed through a form interconnection. This object can be accessed in order to read values 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 it is started and can be accessed by any form in that application. They could have been entered by the user, or they could have been 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 may include wildcards but not comparison operators. Comparisons can be set with a system function. QC objects will 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 invoking them through event rules.
VA	Event rules variables. These objects represent any variables set up by the developer in event rules. They are not manipulated by the system.
SV	System variables. These objects represent some environment variables that have been made accessible to event rules.
SL	System literals. These objects represent some constant system values that have been made accessible to event rules.
TP	Tab Page Object
TK	A column in the table that contains the table event rule.
CO	A constant, for example 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

You can assign values to Query by Example (QBE) Columns using event rules. This allows automatic tailoring of searches to certain situations without adding filter form controls. It also gives the added flexibility of allowing changes to comparison operators at runtime instead of design time like filters.

Processing Available Objects

When the value of an available object is changed during event rule processing:

- 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).
- Then, 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 means:

- FC data and BC data are always identical.
- Whenever FC runtime data is changed, any reference to BC will reflect 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 is moved to the screen.
- On Control is Exited processing, the value entered into the form control is captured in the runtime data structure shared by BC and FC.

Control is Exited Processing

Control is Exited processing includes the following:

- The value in the control is saved to the internal runtime structure(s).
- The *Control is Exited* event is processed.
- If the value has changed since the last time the control was exited:
 - The *Control is Exited and Changed - Inline* event is processed
 - The asynchronous process is launched (the next three steps are executed on a thread)
 - The *Control is Exited and Changed - Async* event is processed
 - Data Dictionary validation is processed
 - The form control data is formatted and copied back to the control

This logic is also performed for each control on the OK Button (Fix/Inspect and Transaction forms).

Grid Processing

You can use system functions to manipulate the selection or sequencing in a grid. This gives you direct access to the database APIs and the selection and sequencing structures used 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 very similar to Set Selection.
Clear Selection	Removes all system-function-defined selection information.
Set Selection Append Flag	Determines whether the system-function-defined information will append or replace QBE and filter field information.
Set Sequencing	Creates one element in the sort structure.
Clear Sequencing	Removes all system-function-defined sequencing information.

Refer to the grid system functions in the *Online APIs* for more information about these system functions.

If a Headerless Detail form contains at least one equal filter or one nonfilter database form control, the grid records for the form are only updated if they are changed.

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 allows you to fetch only a page of data at a time to limit the number of records possible. This is particularly useful for grids where you may be fetching thousands of records.

Grid columns have type ahead functionality. Once a user enters a character in the field, a history list is searched for a match. If there is a match, it is displayed in the field with highlighted text. This 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 whether or not you add event rule logic. However, the engine pauses at specific events, such as *Dialog Is Initialized*, so you can add logic or manipulate values. How events execute is based on event rule logic and user interaction.

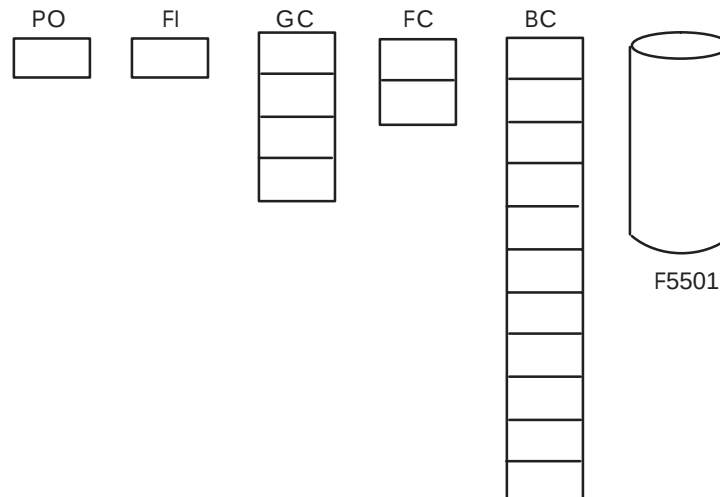
Some of these events occur on each form, although the order and number of events varies depending on the form type. For example, *Dialog Is Initialized* is a good place to add logic on all forms. *Post Dialog is Initialized* is a good place to add logic on a Fix/Inspect form, but not on a Find/Browse form. *Dialog is Initialized* and *Post Dialog is Initialized* for a form with a grid are different than for a form without a grid. Forms with an updateable grid do a find and read the database between these two events.

For more information about the process flow for different form types refer to the Form Processing appendix in this guide.

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.

Address Number	Alpha Name	Long Address	Industry Class	Sch Typ	Tax ID



The runtime engine processes events in a certain order. The typical events for the Find/Browse form and the order in which they are processed follow. This flow can vary depending on user interaction and event rule logic used.

Pre Dialog Is Initialized

The following steps happen before the *Dialog is Initialized* event is processed and the form appears.

- Initialize runtime structures (or memory is cleared)

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
- Filter controls are used to build a WHERE clause of an SQL SELECT statement

- If there was a form interconnection to the form, there would be something in the FI structure in memory. The FI value is copied to the BC runtime structure.

Dialog Is Initialized

The engine pauses for the event to be processed. All event rule logic attached to the *Dialog Is Initialized* event is processed. When the engine pauses, the runtime structures have the following values:

BC=Any FI values passed

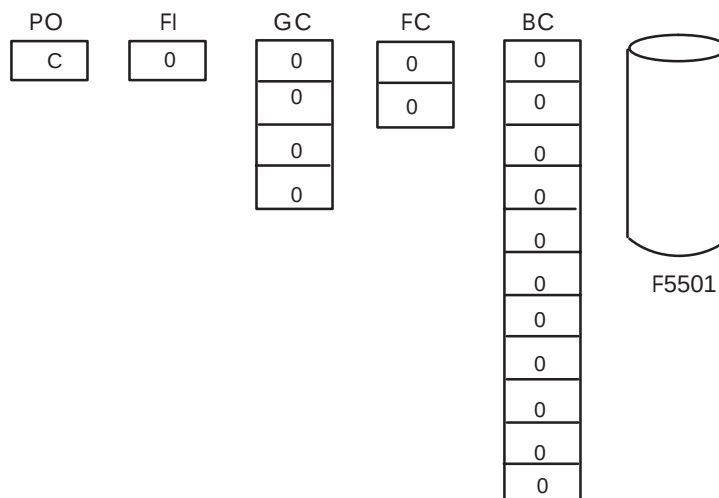
FC=Any FI values passed

GC=0

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

PO=Values passed from processing options

The following illustration represents what is in the runtime structures just before *Dialog Is Initialized* is run.



The *Dialog is Initialized* event is commonly used to:

- Hide or show controls on a form
- Disable controls

After *Dialog Is Initialized* is processed, the form appears.

Post Dialog Is Initialized

The engine pauses again for the event to be processed. You can add logic to be processed when *Post Dialog Is Initialized* is run. When the engine pauses, the runtime structures have the following values:

BC=0 (or values already passed in)

FC=0 (or values already passed in)

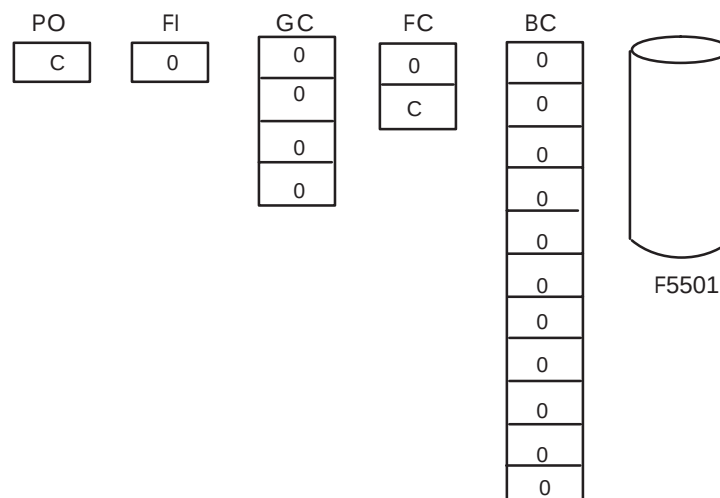
GC=0 (or values already passed in)

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

PO=Values passed from processing options

At this point, no records have been fetched yet. The engine pauses just before the FETCH to allow you to add logic or manipulate the runtime structure values. You can also do an assign from the filter field here.

The following illustration represents what is in the runtime structures just before *Form Record is Fetched* is run.



The *Post Dialog is Initialized* event is commonly used to:

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

This event runs even when a record does not exist.

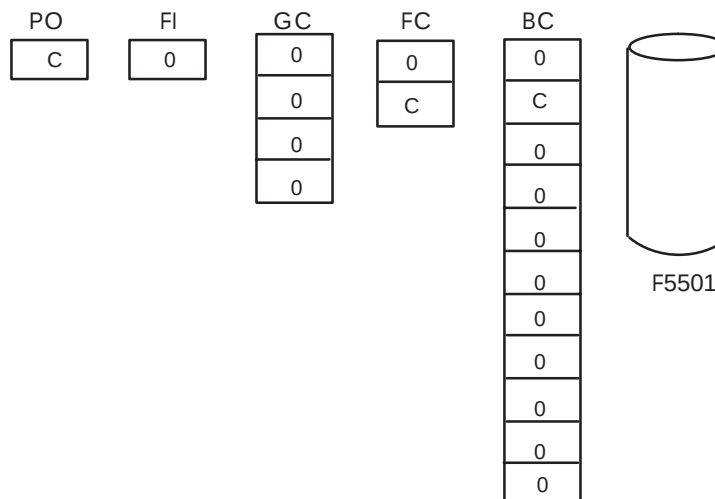
At this point, the user must click the Find button to populate the grid. (The J.D. Edwards standard is to disable autofind.) The SQL SELECT is now built.

SQL SELECT

After the user clicks the Find button, a SELECT statement with a WHERE clause is built.

- The SQL SELECT statement marks all columns in the business view.
- The WHERE clause is built from 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 what is in the runtime structures just before the SQL SELECT is built and the FETCH occurs.



Do in While Loop

Records are fetched one at a time in a WHILE loop.

- WHILE database records are found and WHILE grid rows are available on the form.
- Page-at-a-time processing is done.
- SQL FETCH gets one record

Page-At-A-Time Processing

Page-at-a-time processing means that only the number of records that a grid can visibly display are fetched, for example if the grid only displays three rows, then three records are fetched one after the other. In order 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 are designing 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. This does not cause a new FETCH. For example, if records 45 - 47 are loaded and the user clicks the down arrow, records 48 - 50 are loaded. 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.

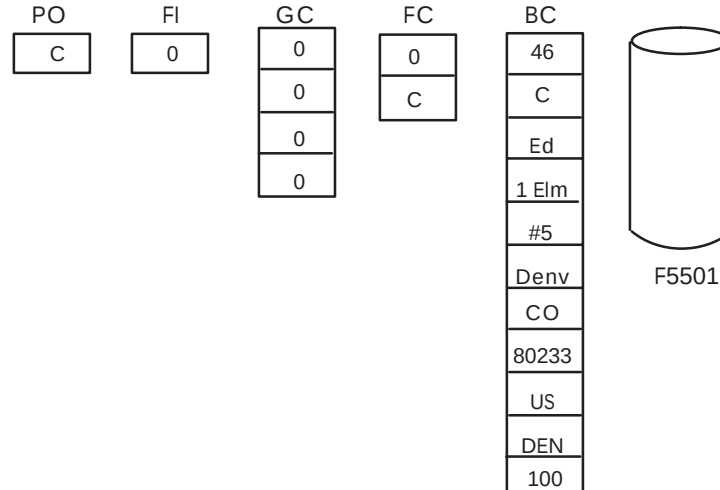
If page-at-a-time processing is on, event rule logic attached to the *Last Grid Rec Has Been Fetch* event will not be executed if the user has not clicked the down arrow key on the grid scroll bar enough to fetch the last record.

Page-at-a-time processing is done for performance reasons. It can be disabled, but the J.D. Edwards standard is not to disable it unless there is a valid business reason to do so or the form type is headerless detail.

BC Assigned Database Values

After the first record in the WHILE loop is fetched, the database values are copied into the BC runtime structure. Values from each marked column in the table are placed in the BC runtime structure elements.

The following illustration represents what is in the runtime structures when the first record in the WHILE loop is read.



Grid Rec Is Fetched

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

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

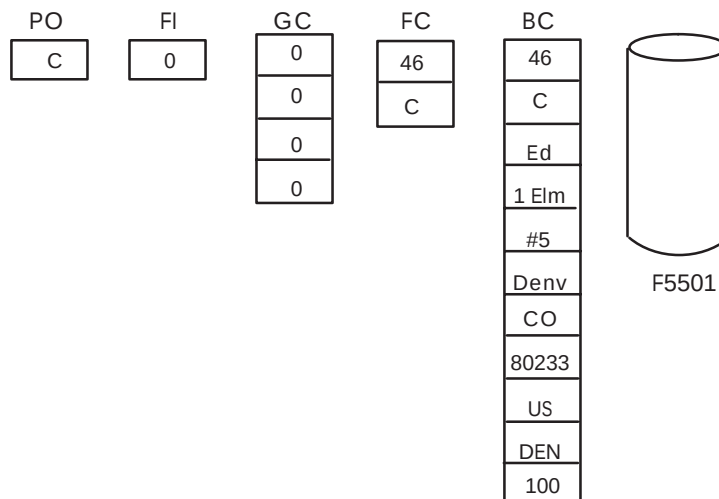
FC=Values from the database (if database fields)

GC=0

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

PO=Values passed from processing options

The following illustration represents what is in the runtime structures just before *Grid Rec Is Fetched* is run.

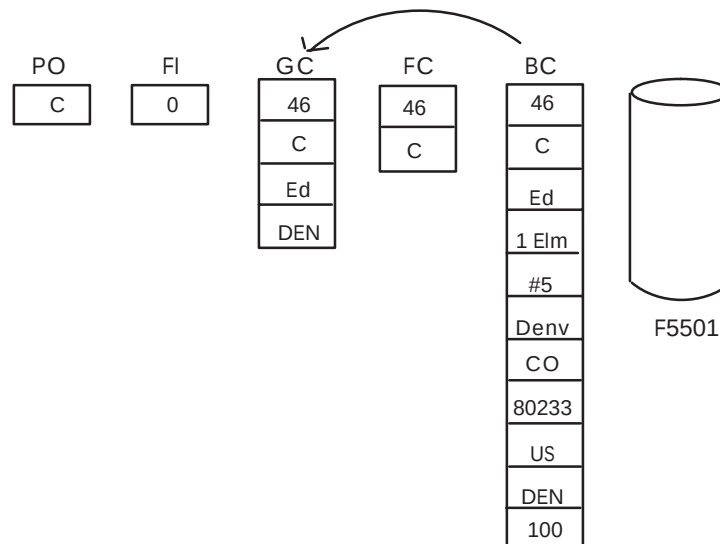


The *Grid Rec Is Fetched* event is commonly used to:

- 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 what is in the runtime structures when the first record is read in a WHILE loop.



Write Grid Line Before

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

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

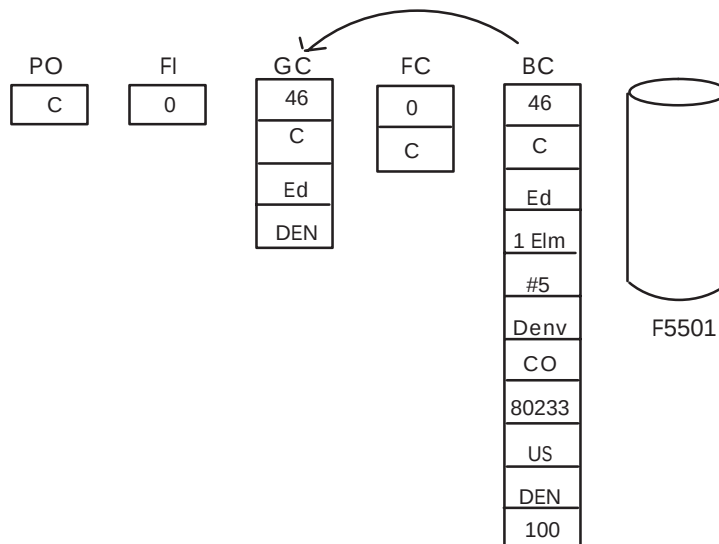
FC=Values from the database (if database fields)

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 what is in the runtime structures just before *Write Grid Line Before* is run.

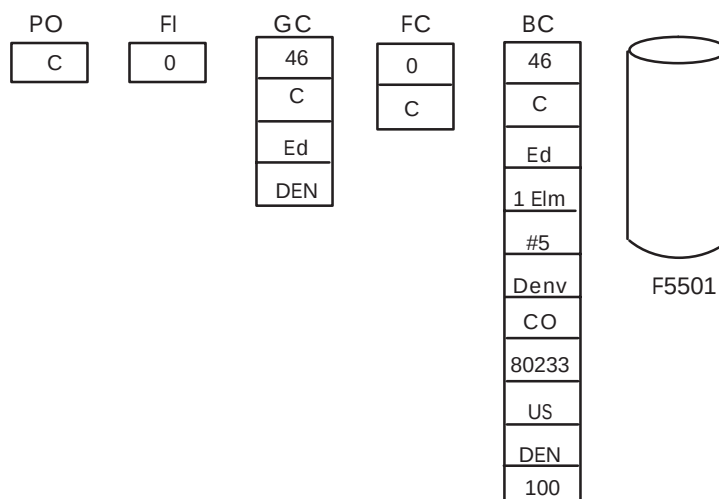


The *Write Grid Line Before* event is commonly used to:

- 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 of any grid value, for example, unit of measure conversion
- Retrieve additional information for the grid row from tables not in the business view. For example, a description.

After the *Write Grid Line Before* is processed, 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 what is in the runtime structures now.



Write Grid Line After

The engine pauses again for the event to be processed. You can add logic to be processed when *Write Grid Line After* is run. 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 database field)

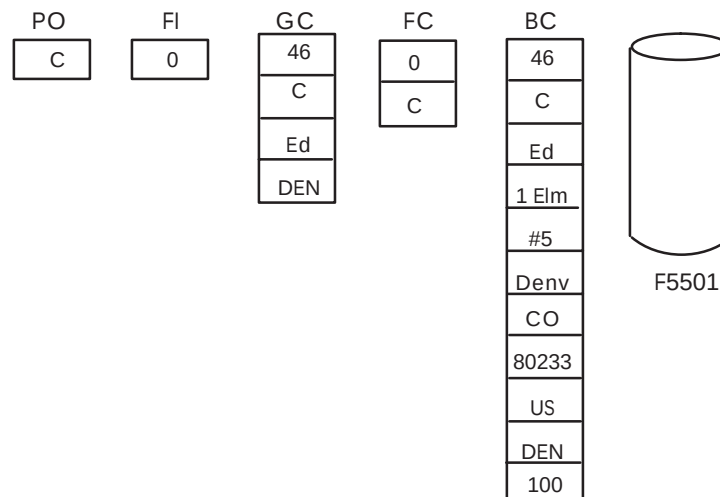
GC=Values from the database (from the first record read)

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

PO=Values passed from processing options

The screen grid cells display the current record.

The following illustration represents what is in the runtime structures just before *Write Grid Line After* is run.



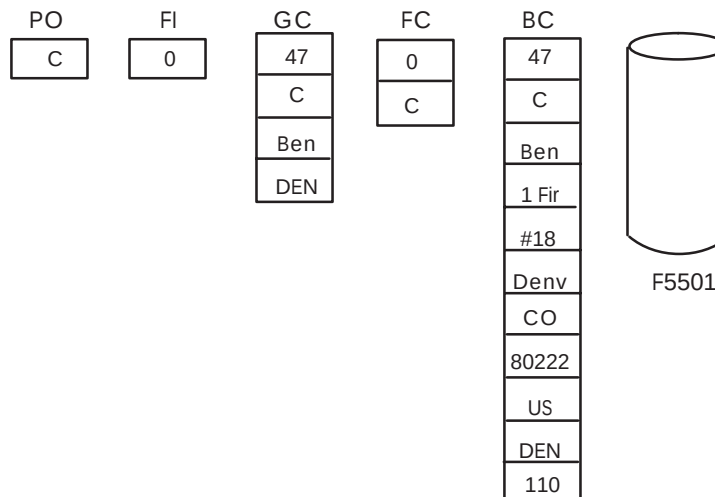
The *Write Grid Line After* event is commonly used to add logic after the user sees a row on the form.

Each record in the database is read one at a time and passes through the same processing steps. When the next record is read the same processing steps occur:

- SQL Fetch of next record that matches criteria
- BC assigned values from the database
- *Grid Rec is Fetched* is processed
- GC is assigned BC values
- *Write Grid Line Before* is processed

- Values appear in the grid row on the screen
- *Write Grid Line After* is processed

The following illustration represents what is in the runtime structures after the *Write Grid Line After* event processes for the second record.



If grid records were found and the user has clicked the arrow keys to fetch all records that meet the selection criteria, the SQL FETCH will finally fail. If the grid is full and the user has never clicked the down arrow key, FETCH may not fail because there may be matching records that have not been read in. When FETCH fails, *Last Grid Rec Has Been Read* is processed.

Last Grid Rec Has Been Read

The engine pauses again for the event to be processed. You can add logic to be processed when *Last Grid Rec Has Been Read* is run. 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 database field)

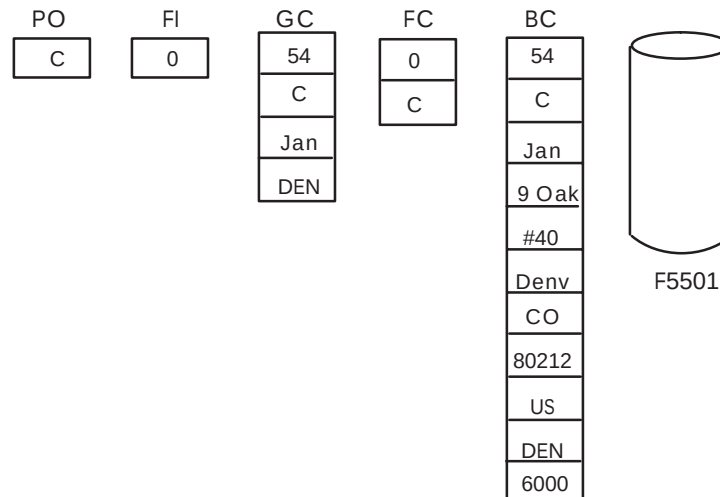
GC=Values from the database (from the last record read)

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 what is in the runtime structures just before *Last Grid Rec Has Been Read* is run.



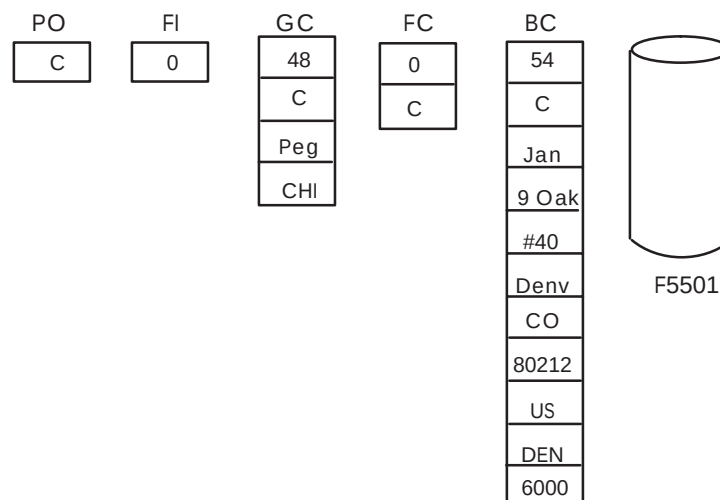
The *Last Grid Rec Has Been Read* event is commonly used to:

- Write total lines to the grid
- Display on the form totals based on grid values

Select Button Processing

When a user chooses a grid row and clicks the Select button, form values are copied back into the GC structure. The BC structure stays the same, and no database updates are made just because the user clicked a row. This is why you should select GC as the value to pass during an interconnection.

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



Button Clicked

The engine pauses for the event to be processed. You can add logic to be processed when *Button Clicked* is run. 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 database field)

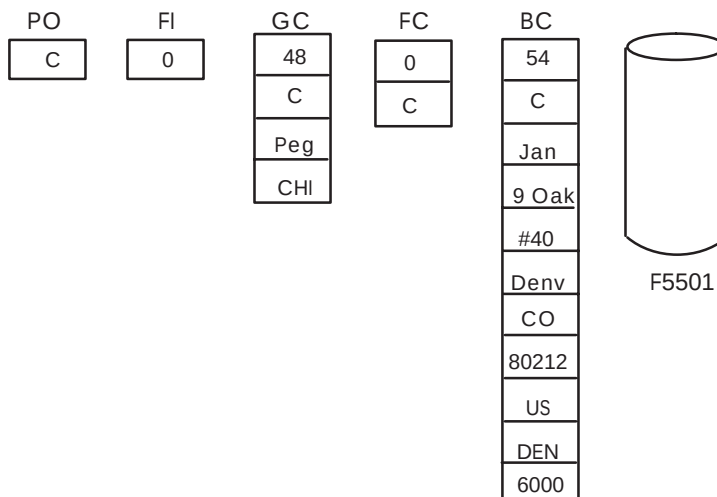
GC=Values from the selected grid row

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 what is in the runtime structures just before *Button Clicked* is run.



The *Select Button Clicked* event is commonly used to:

- Interconnect 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 turn on the form property “Multiple Select” for multi-select to work.

Repeat Business Rules for Grid must be turned on if you want to allow a user to select multiple grid rows.

Add Button Processing

After the user chooses a grid row, the GC runtime structure is assigned the values that show in the grid record on the form. Normally the user would not choose a row before an add, but if a row is highlighted, form values are still copied to the GC runtime structure. If the user does not choose a record, the GC runtime structure will contain the values from the last grid record read in, if any. Database updates are not made 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 *Button Clicked* is run. 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 database field)

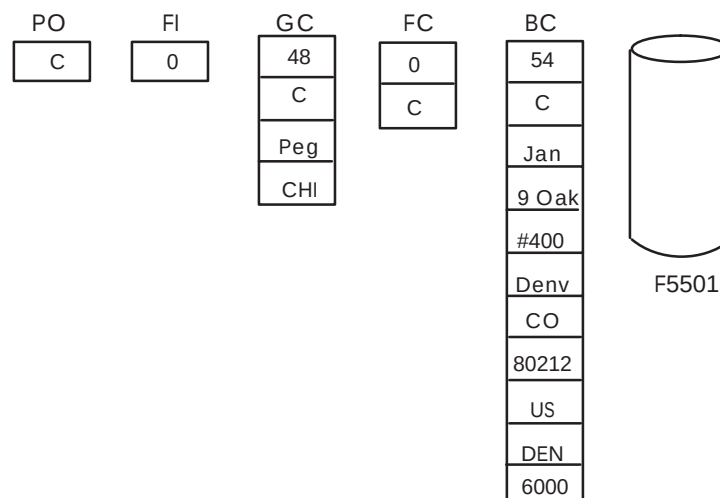
GC=Values from the selected grid row

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

PO=Values passed from processing options

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

The following illustration represents what is in the runtime structures just before *Add Button Clicked* is run.



The *Add Button Clicked* event is commonly used to interconnect to another form, such as a Fix/Inspect or Headerless Detail where the add will actually be performed.

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

Delete Button Processing

When the user chooses a grid row, the GC runtime structure is assigned the values that show in the grid record on the form. BC is still the same, and no database updates have been made.

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

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

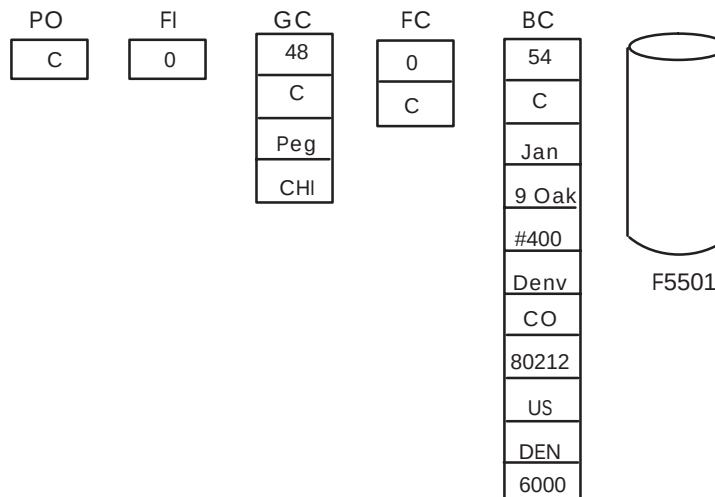
FC=Values from the database (if database field)

GC=Values from the selected grid row

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

PO=Values passed from processing options

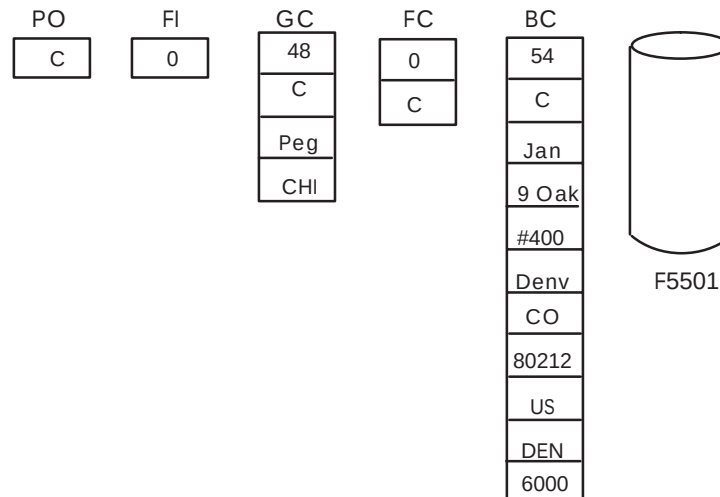
The following illustration represents what is in the runtime structures just before *Button Clicked* is run.



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 *Delete Grid Rec Verify Before* is run. Logic attached to this event is processed after the user clicks Delete, but before the pop-up Verify window appears.



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 *Delete Grid Rec Verify After* is run.

This is where you might want to perform any editing to verify that the delete is valid. For example, there may be some dependent records on other tables that require that this record is not deleted as long as they exist.

The engine pauses for the *Delete Grid Rec Verify Before* event to be processed. You can add logic to be processed when *Delete Grid Rec Verify Before* is run.

Logic attached to this event is processed after the user clicks OK in the Verify confirmation window. If the verify is canceled, the logic attached to this event will 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 *Delete Grid Rec From DB Before* is run. When the engine pauses, the runtime structures FC is blank.

Logic attached to the *Delete Grid Rec From DB Before* event is processed after the user clicks OK to delete and after the Verify window is processed, but before the record is actually deleted.

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

After the *Delete Grid Rec From DB Before* event is processed, an SQL DELETE statement is built. The engine pauses again so you can add logic to process at this point. FC is blank. The SQL DELETE happens at this point, and the current record is deleted from the database. When multiple records are chosen, one delete call can delete multiple 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 *Delete Grid Rec From DB After* is run. This logic will be run after the record is physically deleted from the database.

Logic attached to the *Delete Grid Rec From DB After* event is processed after the user clicks OK to delete and after the Verify window is processed, and after the record is actually deleted.

You might use this event to call a business function to perform deletes on related tables that are not in the business view.

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 *All Grid Recs Deleted from DB* is run. At this point FC is blank.

You can also add logic after the record is physically deleted from the database. Logic attached to the *All Grid Recs Deleted from DB* event is processed after multiple grid lines and the corresponding database records have been deleted. Rules attached to this event are not apparent on the screen 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, suppose you want to pass data for a selected record on a Find/Browse form to a Fix/Inspect form to make revisions to that record. To accomplish this task, you would create a form interconnect event rule and attach it to the Select toolbar option for the event, Button Clicked.

This chapter describes the following:

- ☐ Creating and saving event rules
- ☐ Adding comment lines to event rules
- ☐ Printing event rule logic
- ☐ Validating event rules

Before You Begin

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. Should logic occur when:

- ☐ Initializing the form?
- ☐ Clicking a button?
- ☐ Exiting field?
- ☐ When a row is changed or exited?

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, a form is also a control, and you can create logic that automatically processes whenever a form is initialized.

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 displays for creating and manipulating the event rule line by line.

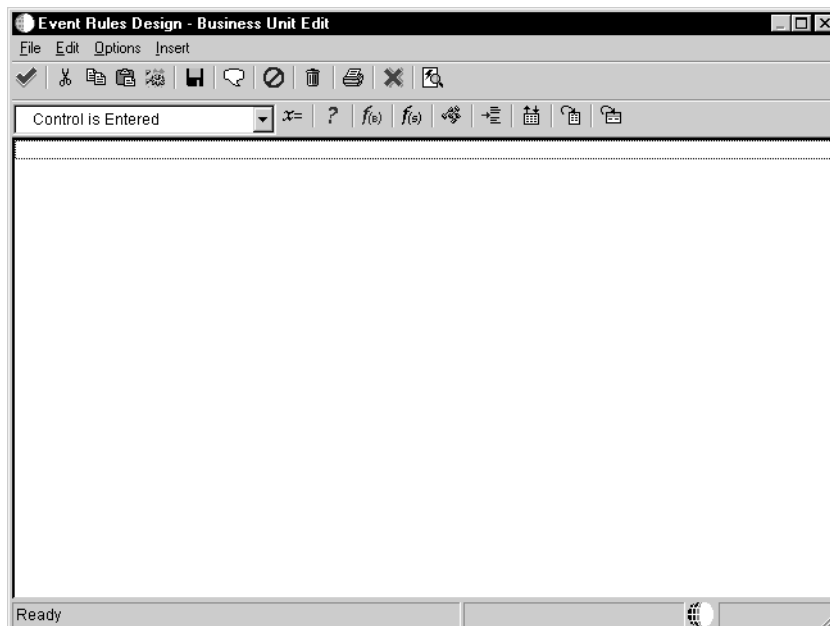
Click on a specific button within Event Rules Design to:

- 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

On Forms Design

1. Choose a control on the form.
2. From the Edit menu, choose Event Rules.



3. Choose an event from the Events list box.
4. Click the appropriate Event Rules button to add logic for the control:
 - Business Function
 - System Function

- If/While
 - Assign
 - Forms
 - Reports
 - Else
 - Variables
 - Table I/O
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 only affects 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, hold down the shift key and select all event rules you wish to enable, disable, or delete. After you select the desired lines, click the Enable/Disable or Delete button.

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

Field	Explanation
Disable Event	Disables (but does not delete) the event rule for a control. This is useful in debugging your applications. You can disable one or more rules for a specific control until the problem is identified. Disabled event rules are marked with a red ! in the event list.
Repeat Business Rules for Grid – ER Dsn	Applies an event rule to each of the grid selections. This check box is available only for grid controls. Note: Not all grids allow multiple selections. The developer must specify the Multiple Selection option in Grid Properties form.

Field	Explanation
Save	Saves all rules for all events associated with the selected control. Note: The OK button also performs a save before exiting.
Comment - Event Rules Design	Inserts a comment into the event rule. Comments do not affect logic. Use comments to document the business rules.
Enable/Disable - Event Rules Design	Enables or disables a single line of the event rule. Disabled event rule lines are marked with a red exclamation (!).
Delete	Destroys an object or record. On Event Rules Design, the Delete button removes a selected line, one at a time. When an IF/WHILE statement is deleted, the associated ELSE and END clauses are also deleted, but the Rules inside those statements are not deleted.

Finding Event Rules

You can search for strings in event rules.



To find a string of text in an event rule

From Event Rules Design

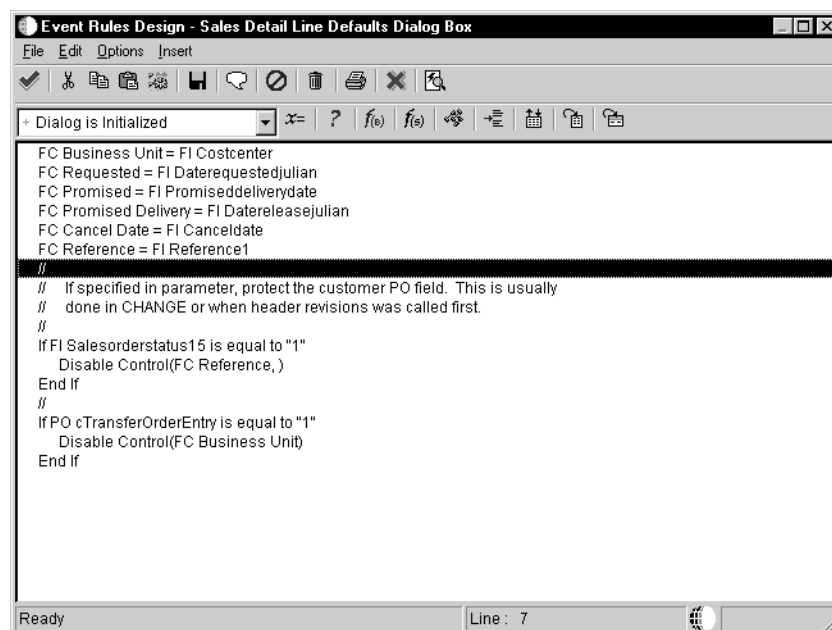
1. From the Edit menu, Choose Find.
2. Indicate the direction you want to search.
3. Type the string of text you want to find and click the Find Next button.



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 documentation purposes.



You can use any of the following buttons:

- Cut
- Copy
- Paste
- Paste Options

When you cut event rules, the selected lines are deleted from the source and stored on the clipboard.

To cut event rules

On Event Rules Design

1. Select one or more lines of event rules.
2. Click the Cut button.

To copy event rules

On Event Rules Design

1. Select one or more lines of event rules.
2. Click the Copy button.

When you copy event rules, the selected lines of event rules are copied from the source and stored on the clipboard.

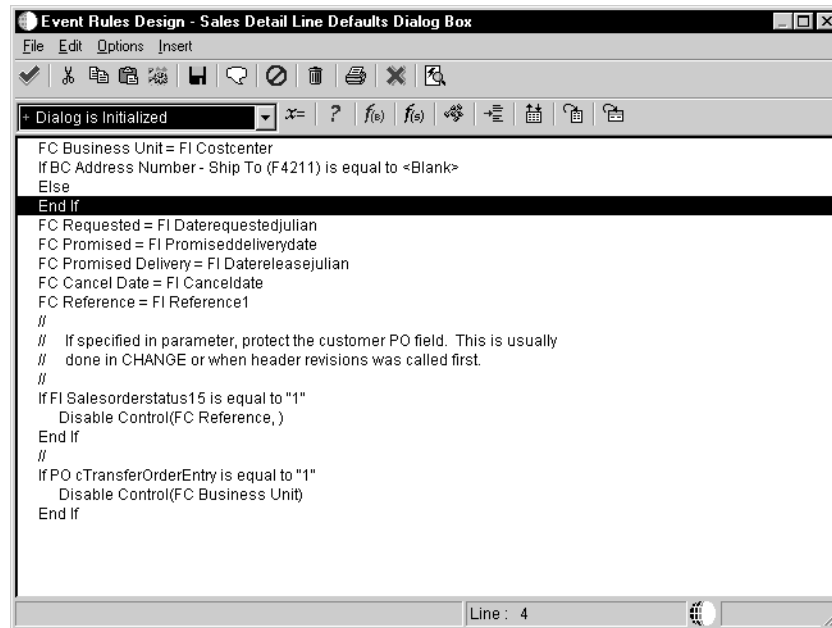
To paste event rules

On Event Rules Design

1. Select the line after which you wish to insert your lines of event rules.
2. Click the Paste button.

When you paste event rules, objects from the source are resolved as they are pasted. If an object is partially resolved, the closest matching object from the destination event rules will be pasted. A comment line appears above the partially resolved line of event rules and in the status bar to let you know there is a partially resolved object.

Some objects cannot be resolved in the destination event rules. These lines of event rules will be disabled and a comment will appear. For example an EndIf statement is commented out if its associated If statement is missing.

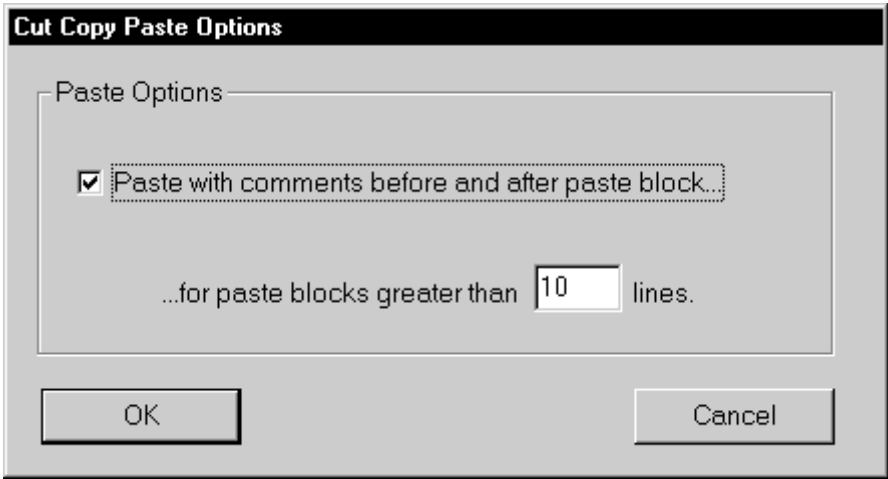


For criteria statements, the paste operation adds whatever is necessary to maintain a clean logical structure. For example, if you paste an If statement and there is no EndIf statement, the paste operation will add a matching EndIf statement to make the logic complete.

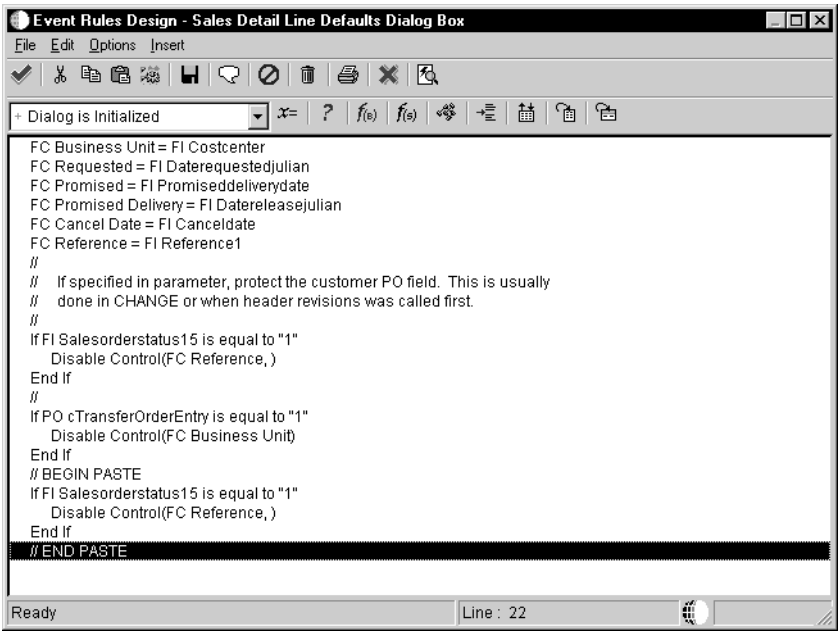
You can set paste options to display comments before and after a block of pasted event rules.

► To set paste options

1. Click the Paste Options button.
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 will appear to let you know where your paste begins and ends.



Field	Explanation
Paste with comments before and after paste block	When this option is checked, comments are inserted before and after the section of event rule logic.

Adding Comment Lines to Event Rules

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

► To add a comment line to event rule

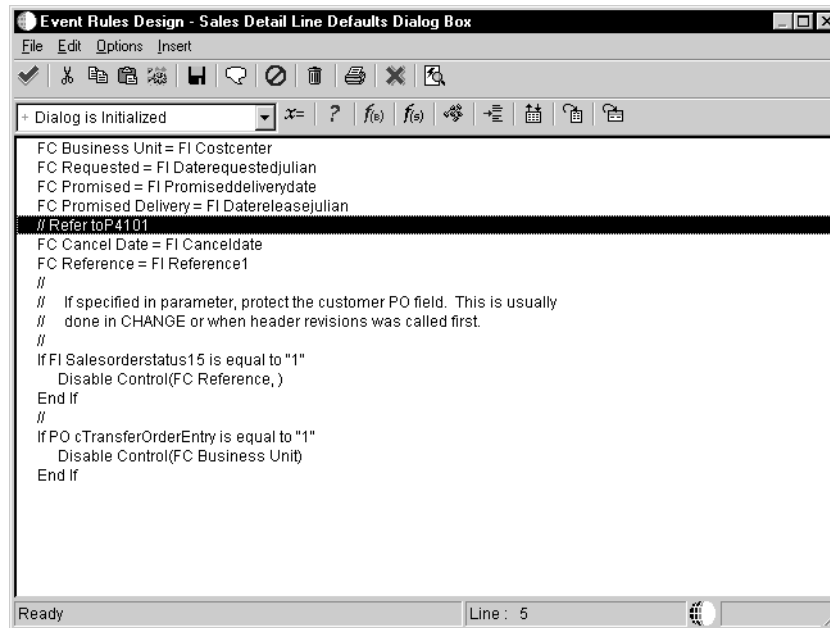
On Event Rules Design

1. Position the cursor where you want to add a comment.
2. Click the Comment button.



3. Type a comment in the input area and click OK.

The comment is automatically formatted in Visual C++ notation so you do not have to type the `//` to indicate a comment.



Printing Event Rule Logic

You can print the code for event rules. This is especially useful when there are many lines of code that comprise an event rule. You can print event rule code for:

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

► To print event rule code

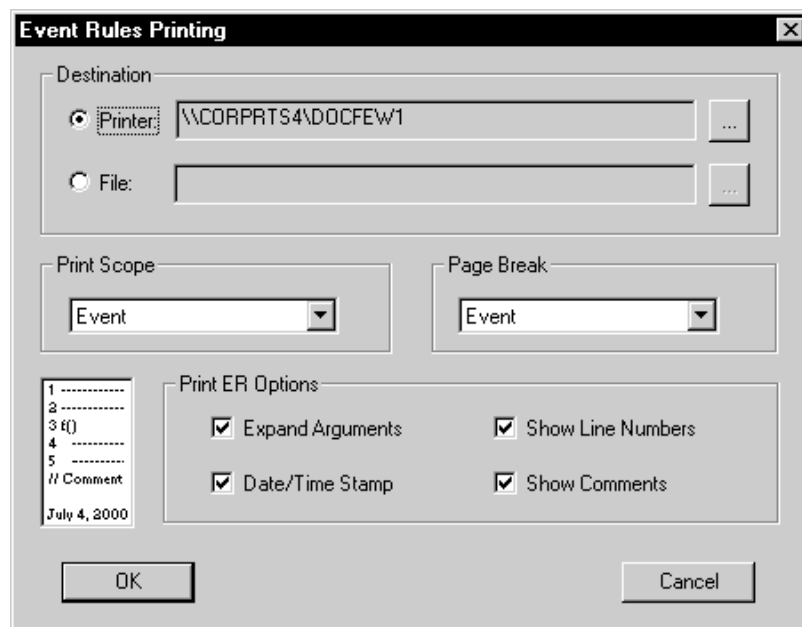
On Form Design

1. Click on 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 displays in the work area.

4. From the File menu, choose Print.

5. On Event Rule 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 argument
 - Date/Time Stamp
 - Show Line Numbers
 - Show Comments



Field	Explanation
Print Scope	The object you want to print for the event rule. You can print the logic for the application, the form, a specific control, or an event. This is useful when you don't need to print out the logic for the entire application and only want to review a specific portion of the event rule logic. You options are: • Application - to print event rule logic for form controls across the entire application • Form - to print event rule logic for form controls on this form • Control - to print event rule logic for all events for the control, where logic has been built • Event - to print event rule logic only for the selected event on the control
Page Break	Begins printing a new page of logic at a specific level of code in the event rule. You must specify where the break occurs, otherwise the default is to print out a page for each event.
Print ER Options	You can include additional information in the print out of event rule logic, such as detail for each argument, the date and time of the print out, line numbers for the event rule, and comments that appear in the logic.

Validating Event Rules

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

The event rules are validated when you select Save or when you select Exit and save before exiting. You can exit Forms Design Aid or leave it open if errors are found during validation upon exit.

- If you select Validate Event Rules Now, event rules will be validated immediately.



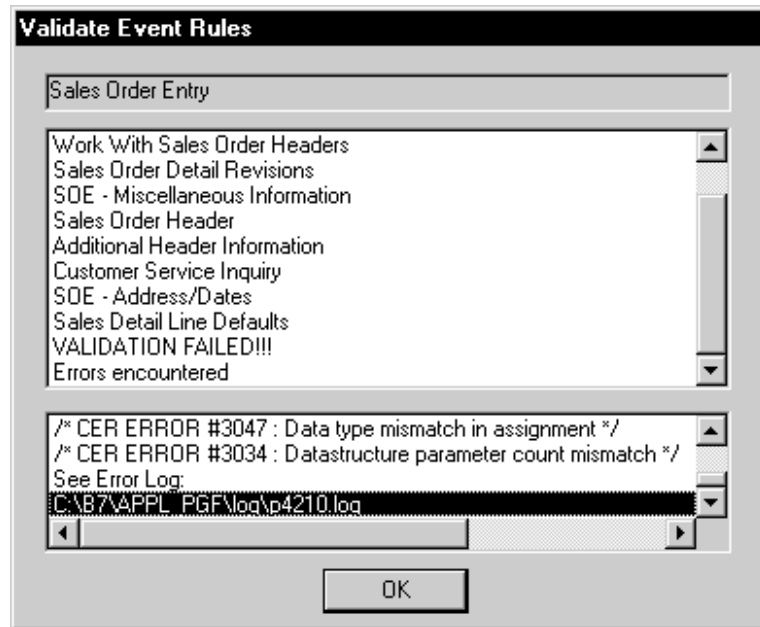
To validate event rules

On Forms Design

1. From the Application menu select:
 - Validate Event Rules Now

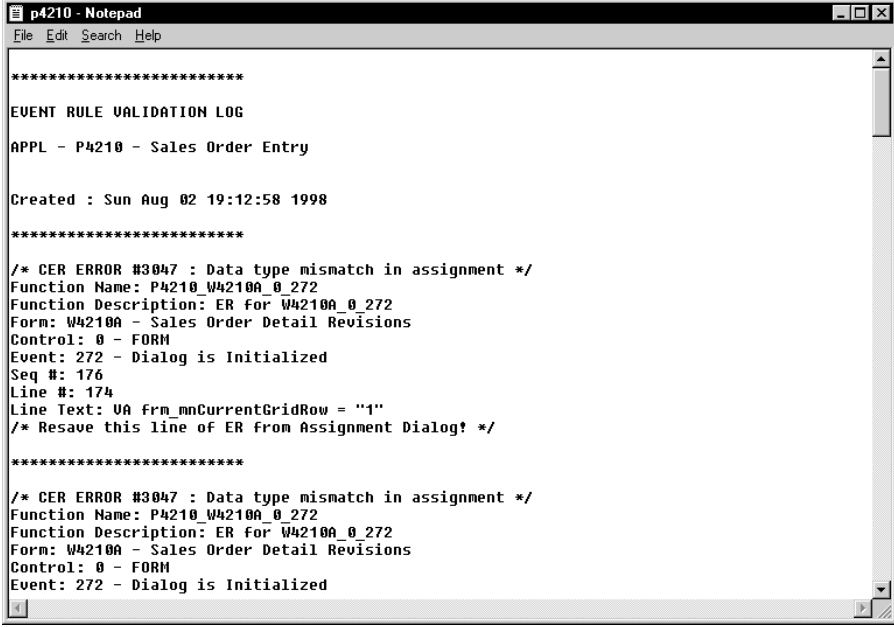
The Validate Event Rules form displays messages as it processes. When validation has successfully completed a generation successful message appears.

The following example illustrates what the Validate Event Rules form displays when it finds an error.



Validate Event Rules checks for errors at two levels. The first level is in Business Function event rules or Table event rules where the event rule code is converted from the event rule scripting language to C code and then into compiled code. The second level is at the application level where the event rules are checked 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 created is stored in a file such as `b7\prod\log\p1234.log` (where `prod` is your environment). If there are no errors, no log is generated.



```
p4210 - Notepad
File Edit Search Help

*****
EVENT RULE VALIDATION LOG
APPL - P4210 - Sales Order Entry

Created : Sun Aug 02 19:12:58 1998

*****

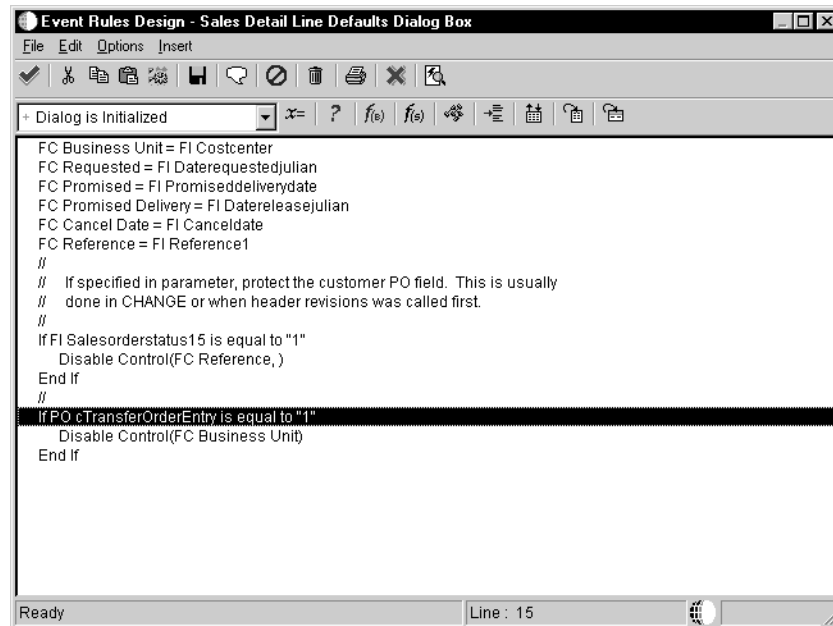
/* CER ERROR #3047 : Data type mismatch in assignment */
Function Name: P4210_W4210A_0_272
Function Description: ER for W4210A_0_272
Form: W4210A - Sales Order Detail Revisions
Control: 0 - FORM
Event: 272 - Dialog is Initialized
Seq #: 176
Line #: 174
Line Text: VA frm_mnCurrentGridRow = "1"
/* Resave this line of ER from Assignment Dialog! */

*****

/* CER ERROR #3047 : Data type mismatch in assignment */
Function Name: P4210_W4210A_0_272
Function Description: ER for W4210A_0_272
Form: W4210A - Sales Order Detail Revisions
Control: 0 - FORM
Event: 272 - Dialog is Initialized
```

Working with If and While Statements

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



If Statements

An If statement executes conditional event rule logic. That is, the logic executes only if the condition is true. An If statement is evaluated only once upon execution of the event rule.

An If statement takes this general form:

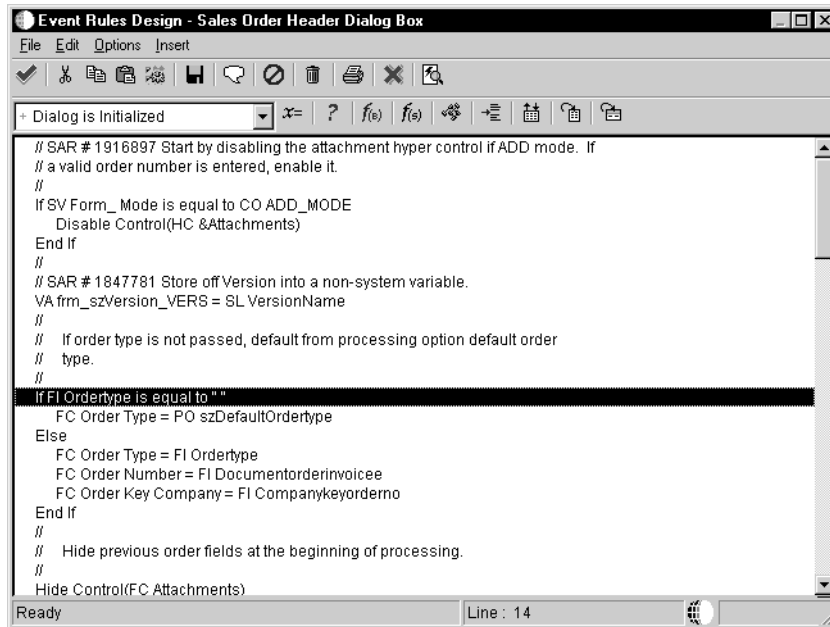
```
IF condition X is true
  Do Y
ELSE
  Do Z
ENDIF
```

An If statement is evaluated as follows:

If condition X is true, then do Y and stop evaluating the statement. If condition X is not true, then go to the Else clause and do Z.

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

The following is an example of an If statement from 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 takes this general form:

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

A While statement is evaluated as follows:

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

Creating If and While Statements

Use the If/While button to build conditional logic into an event.

► **To create an If or a While statement**

On Event Rules Design

1. Select an event in the Event Rules Design window.
2. Click the If/While button.

Operator	Left operand	Comparison	Right operand
If	BC Address Number - Parent (F4201)	is equal to	BC Address Number (F4201)

Each cell in the Criteria Design grid represents a component of the criteria. When you select a cell, by clicking on it or tabbing into it, a list of valid options displays. To select an option either double-click on it or select it, and press enter or tab. You can select an option by using the mouse or by typing in the option you wish to select.

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

3. Choose one of the following Operators:
 - If
 - While
4. Choose a Left Operand from the list of available data items.

You can click the right mouse button to sort the available data items by name or object type. If there is only one type, the sort options are not available.

The available data items can be grouped 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

5. Choose a Comparison from the 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
6. Choose a Right Operand from the object list.
7. To assign a literal, select <Literal>.

The Literal dialog is data item dependent.

See *Defining Literal Values in Event Rule Logic* for detailed instructions on assigning a literal.

Criteria Design

Enter condition by selecting from the options provided in each cell of the template below. You may either use the mouse or type the initial characters to select an option.

	Operator	Left operand	Comparison	Right operand
	If	BC Line Number (F4211)	is equal to	BC Line Number (F4211)

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

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

You can select the And or Or option to continue your logic to create complex if statements.

What You Should Know About

Inserting If statements

When you create an If statement, an Else clause is also automatically inserted. However, it can be deleted using the Delete button and then reinserted using the Insert Else button.

Deleting If or While statements

When an If or While statement is deleted, the associated Else and Endif or Endwhile clauses are also deleted, but the rules inside those statements are not deleted.

Resequencing statements

You can drag and drop statements on a line-by-line basis to change their sequence.

Resequencing event rules can result in improper syntax. When you click the Save button or OK, a syntax check is performed. If syntax errors are detected, you can either:

- Disable the event rule and continue
- Edit the event rule to eliminate the errors

Using Range and List values

“Is Equal To” and “Is Not Equal To” are the only valid logical operators when using range and list values.

The criteria design tool protects you against 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.

Changing a statement

Select the cell/value you want to change by clicking on that cell. You can then change the value of that cell.

- A syntax check is performed when you click OK.
- To move a line (resequence) select the entire row by clicking on the row header. Then use the arrow up/down on the toolbar to move that line.
- You cannot move the If/While line.

Deleting a statement

Select the line of the statement and click Delete.

The top-most clause of the statement (If or While) cannot be deleted in If/While design, it can only be deleted from the Event Rules Design window.

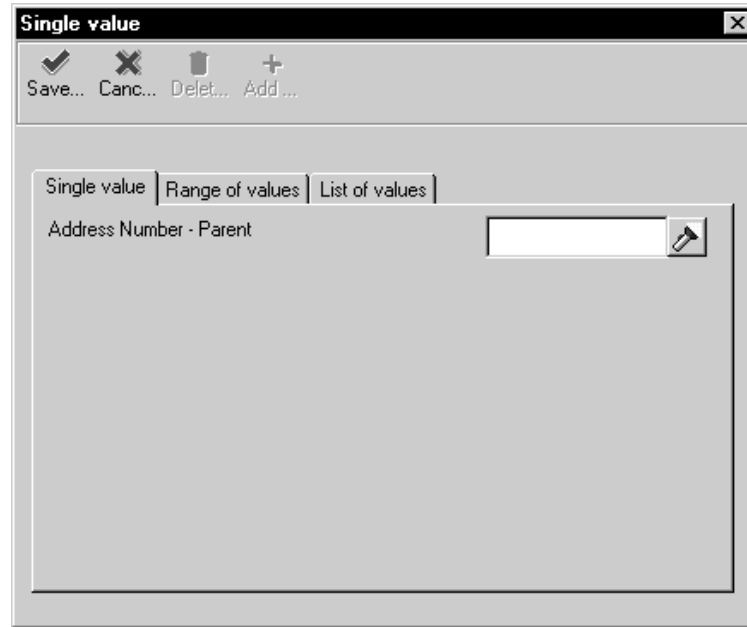
Defining Literal Values in Event Rule Logic

You can define literal values in event rules.

To define literal values

On Criteria Design

1. From the Right operand list, double-click on <Literal>



2. Click on one of the following tabs:
 - Single value (to specify a single value)
 - Range of values (to specify a range of literal values)
 - List of values (to specify a list of literal values)
3. Complete the fields on the form you have chosen.
4. Click Save.

Field	Explanation
Value - Literal	Defines a specific alphanumeric value as a literal in Event Rules Design.
Range - Literal	Defines a range of alphanumeric values to a literal within Event Rules Design.
List - Literal	Adds or removes values from the list to be used in the literal value.

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. Once created, it can only be used in a specific interactive application, batch application, or business function event rule.

The event rule variable is automatically initialized 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 it can be used. For example, you can:

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

Different scope options are available for interactive and batch applications.

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 retain values until the form is closed.

Grid-Level

Grid-level variables are subtypes of form variables, and are available on any form with 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 reinitialized each time a new row is added to the grid. You can use these variables as temporary work fields for the grid. If possible, you should use variables instead of hidden work fields for better performance. Using variables instead of hidden work fields enhances the performance of OneWorld forms during initialization and saves

time because variables are not formatted. Although grid variables are available in all event rule line types, you should only use them with events that are associated with grid rows.

Event-Level

An event-level variable is available only within the event for the form control where it was added. The variable is reinitialized 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 it was added. Use it in event rules just as you would any available object in the list. If you create an event level variable and it is not used in event rules, Form Design Aid automatically deletes it.

Once you add an event rule variable, you cannot be modify it. However, you can delete it.

Variables are automatically assigned 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

On Event Rules Design

1. Click the Variables button.

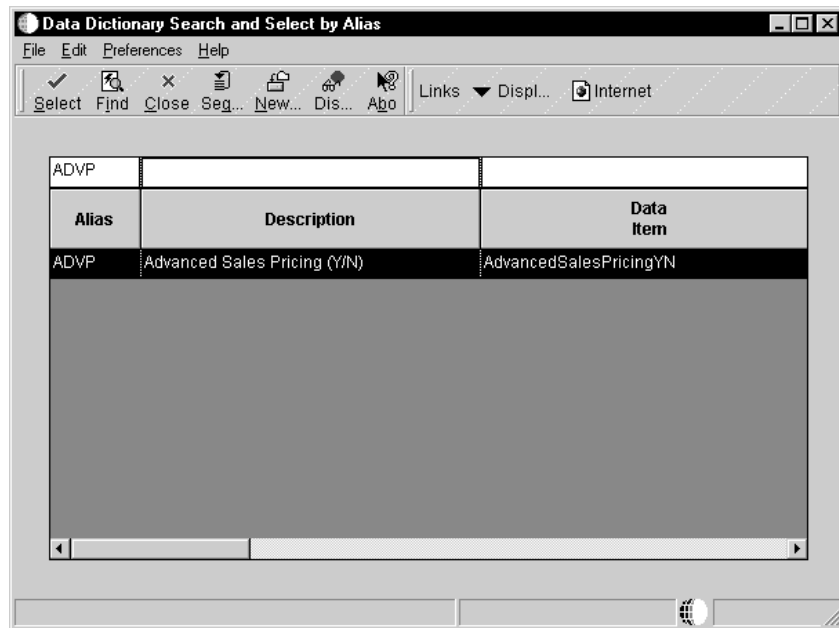
The Variables form displays different scope options depending on whether you are working with an interactive application, batch application, or business function event rule.

The Variables dialog box is shown. It features a list of variables: frm_ADVP_Adv_Pricing, frm_cCurrencyConversionMethod, frm_cPricePickslipYN_PLST, frm_cSpotRateAllowed, frm_DTAI_Error_Message, frm_ERRC_Error_Code, and frm_EV22_cCacheOn. The 'Form' scope is selected. The 'Data Dictionary' section is empty.

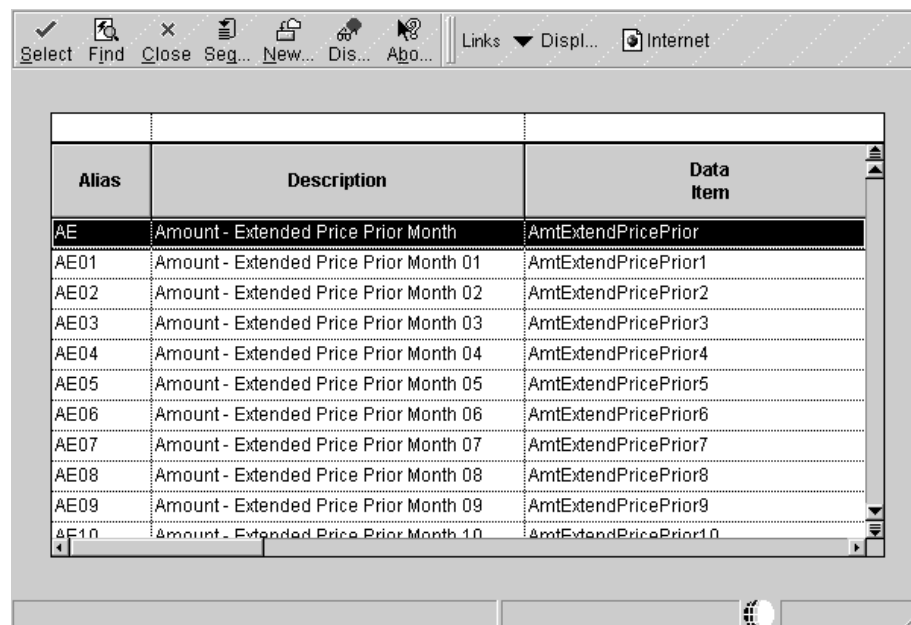
2. Complete the variable naming field.

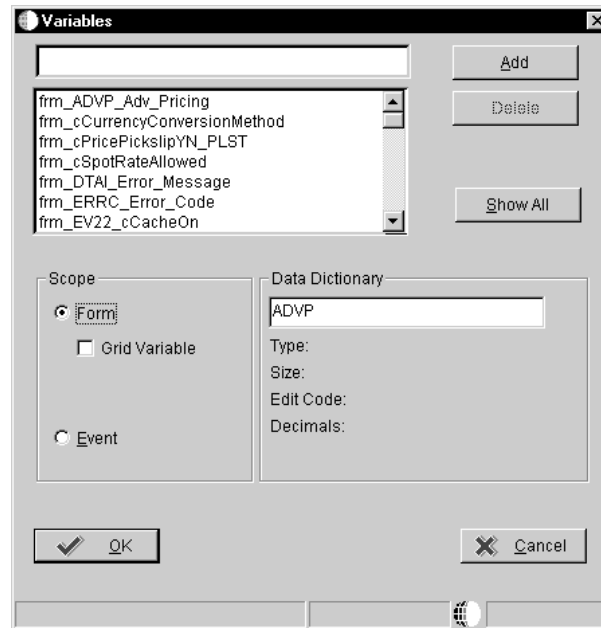
Use standard naming conventions (Hungarian notation). For example, cErrorFlag_E101.

3. Click one of the following Scope options, depending on the purpose for which the variable is created:
 - Form
 - Event
4. Turn on the Grid option if you select Form Scope and you wish to use a grid variable.

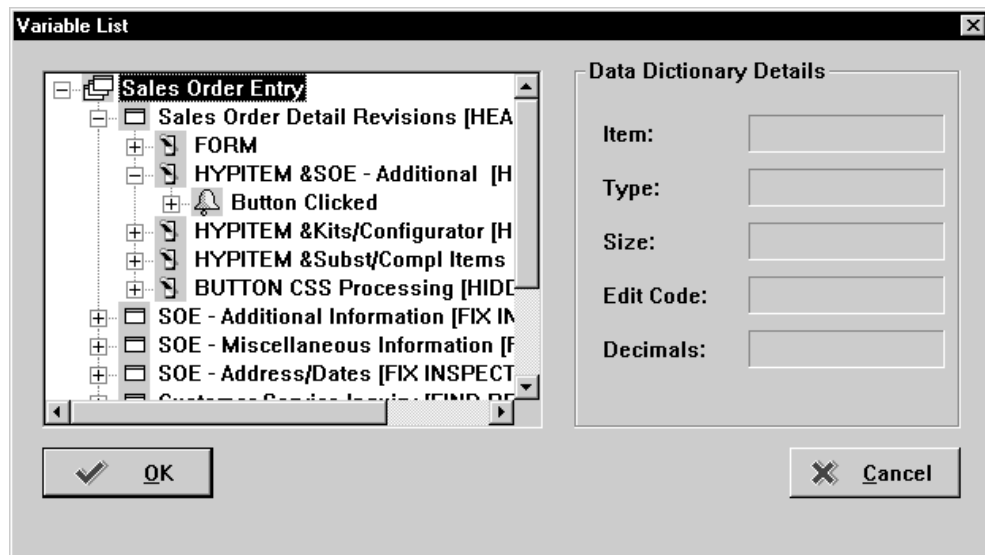


- Click the Data Dictionary visual assist to browse data dictionary items.





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 variable is automatically assigned a prefix based on the type of scope that you choose.

8. Click OK to return to the interactive, batch, or business function event rules design mode and use the newly created variable.

J.D. Edwards Naming Standards

Event Rule variables are named similar to C variables and should be formatted as follows: **xxx_yyzzzzzz_AAAA**

xxx = Depending on the scope, OneWorld automatically assigns the prefix, such as:

frm_ (form scope)

evt_ (event scope)

y = Hungarian Notation for C variables:

c - Character

h - handle request

mn - Math Numeric

sz - String

jd - Julian Date

id - Pointer

zzzzzz = programmer-supplied variable name; capitalize each word

AAAA = Data Dictionary alias (all upper case)

For example, a Branch/Plant event rule variable would be evt_szBranchPlant_MCU.
Do not include any spaces.

Field	Explanation
Scope - Variables	Determines how the variable can be used. For example, you can reference: • A report variable anywhere in the report • An event variable only within the event where it was created Different scope options are available for interactive and batch applications.

Using Event Rule Variables for Automatic Line Numbering

You can use event rules to create automatic line numbering functionality.

► To enable automatic line numbering event rules

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_LineNumber_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 Functions

You can attach the following functions to events:

- System Functions
- Business Functions

Attaching a System Function to an Event

You can use the System Function button to attach predefined J.D. Edwards system functions to events. For example, you can attach system functions to an event that can:

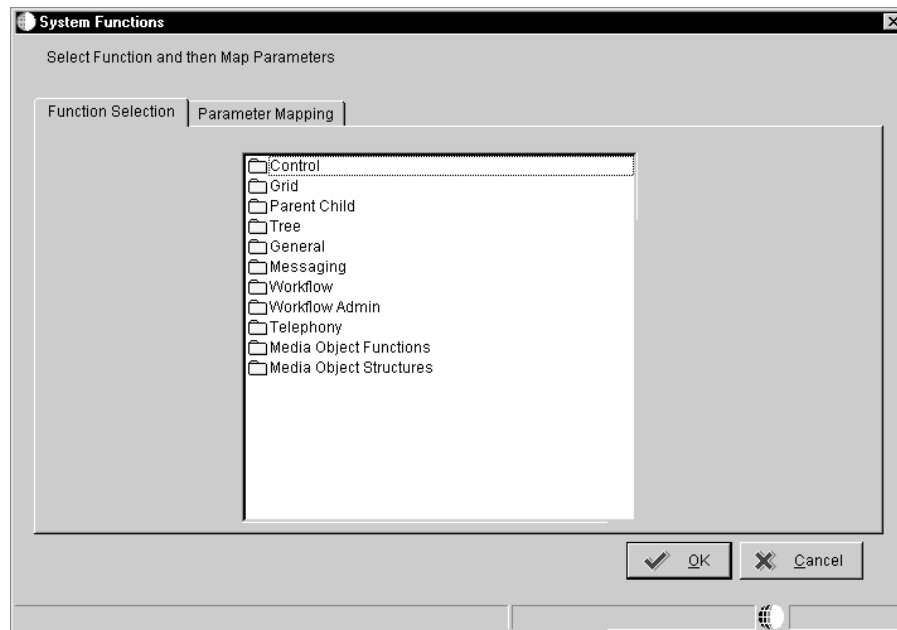
- Hide or display a control
- Insert dates
- Display media objects



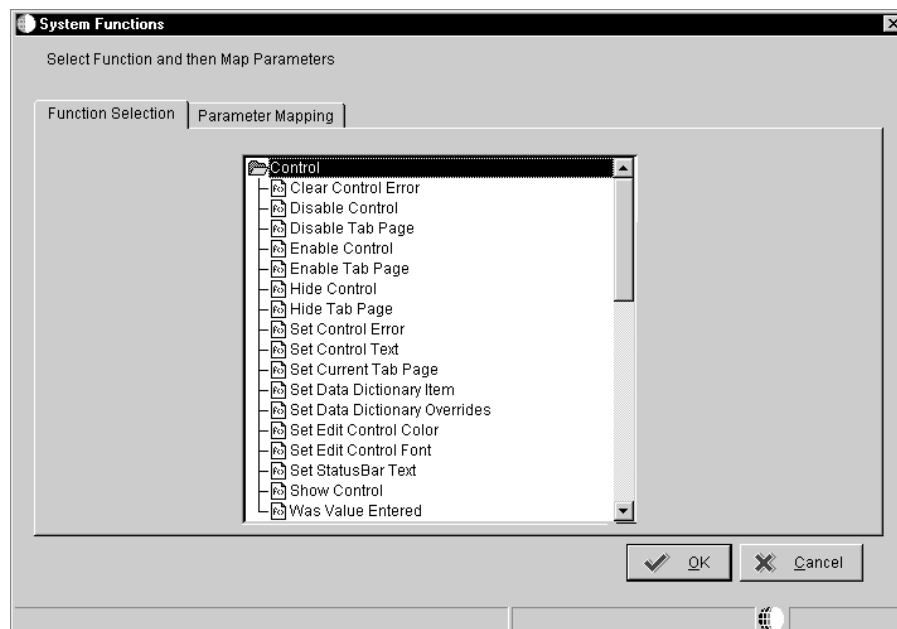
To attach a System Function

On Event Rules Design

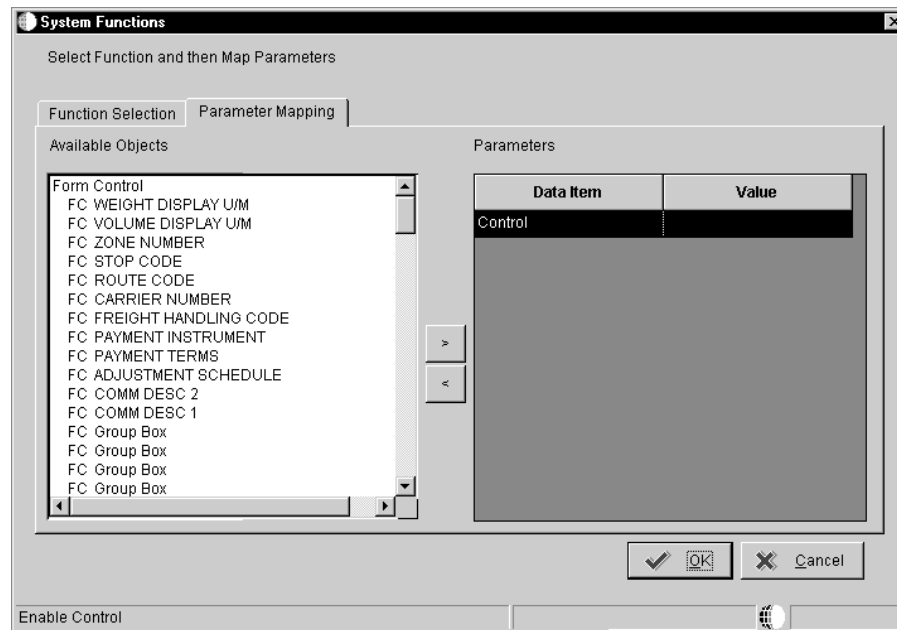
1. Choose an event.
2. Click the $f(S)$ button.
3. Choose a category in the System Functions box.



4. Choose the system function 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 reasons why you might use them. Refer to *System Functions* in the Online APIs for more information about different system functions.

Control System Functions

Clear Edit Control Error

Clears errors that have been set for a specific control. This is important because of the the runtime engine automatically set and clears errors on specific events, but does not clear errors on other events. For example, if someone uses Set Edit Control Error on *Dialog is Initialized*, there is no way for that event to 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. For more information on how the runtime engine sets and clears errors refer to *Messaging* in this guide.

Disable Control

Use this system function if you need 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 someone to change, for example, the exchange rate on a transaction.

Enable Control	You can use this system function to enable a control, for example, if you only want a manager to have the authority to change something.
Hide Control	You can use this system function to dynamically hide and show a control. You might hide a field until the information it holds is specifically requested. For example, in address book, when you add a record you can display the previously added record.
Set Edit Control Error	Sets an error on a specific control and turns the field red.
Set Status Bar Text	Displays text in the status bar. This may be used to describe why an error has occurred and to clear the status bar after an error has been corrected. Another example of when you might want to use this system function is if you have a process running in the background or one that takes some time to complete. You might want to display a "Please Wait ..." message.
Was Value Entered	Verifies that data has been changed, for example if an error occurs and the information is then changed. One of the options for this system function is an All Controls value that will edit if any of the controls have changed. This is useful in transaction type programs.

Set Edit Control Color and Set Edit Control Error are two other control system functions you can use. However, it is a J.D. Edwards standard not to use these. Set Edit Control Color can be used to make mandatory fields a specific color. If you do use this system function, be aware of possible issues regarding users who may be color blind. Set Edit Control Font may be used to draw attention to a field, for example, a negative number, but once again it is a J.D. Edwards standard not to use this because it may cause confusion or detract from the way your form looks.

Message System Functions

Mail Send Message	This system function will return back a message ID (serial number) that is the identifier for the message being created. This is the key parameter for the other messaging functions and the application uses this ID to perform the other system functions. This function might be used to generate a message to a manager for approval for a client to exceed an authorized credit limit.

Forward Message	Forwards a message, either leaving the original in tact, or forwarding the same message on.
Mail Delete Message	Deletes a mail message.
Update Message	Use this to modify a message with new text. You can then use Forward Message to send the new message.

General System Functions

Copy Currency Information	Use this to ensure that when a change is made it is copied throughout all the applications that use it. This would be useful in a country where the decimal place changes frequently. It could also be used if a company changes from the base of one currency to another.
Press Button	You could use this event if you want to reuse event rules from event to event without having to write and maintain the code in all of the places that you use these event rules. To do this you would create a push button on a form and put all of the reusable event rules on the button clicked event. Then you would use this system function to press the button and run the ER from all of the other events where you want to run the ER.
Set Control Focus	Use this to set focus on a specific spot. For example, you could set focus on the detail part of the grid when a user enters a form. This is also useful for overriding the default cursor position, particularly if the focus will be different for and add and a change.
Stop Processing	This could be used to stop processing, for example, if an error occurs.
Suppress Add	This can be used to control the normal flow of the form for I/O purposes. An example of this is transaction entry programs that use master business functions. The master business function does all of the I/O, including adds.
Suppress Delete	This can be used to control the normal flow of the form for I/O purposes. For example, many J.D. Edwards transactions, such as Accounts Receivable and Sales Order, use master business functions to insure referential integrity. You might use this to prevent a user from deleting a customer's address book record that has been used on existing invoices.

Suppress Find	Use this 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 the current record. For those forms that do inquiries, you might want to make sure that the user has provided information that will match an index.
Suppress Update	Use this to suppress an update. This can be used to control the normal flow of the form for I/O purposes.
Was Form Record Fetched	Use this to ensure that a record was actually fetched.

Grid System Functions

Clear Grid Buffer	Use this to clear the buffer. Because the grid buffer is a temporary location 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 then use Clear Grid Buffer to clear the buffer after its use.
Clear Grid Cell Error	This is similar to Clear Grid Buffer.
Clear QBE Column	This is similar to Clear Grid Buffer. An example of when you might use this is if the QBE column had a value forced into it, like the current date on a purchase order Find/Browse form. Then after a Find is executed, you could clear the QBE column in case another date need to be used.
Copy Grid Row to Grid Buffer	You might use this if once Clear Grid Buffer was issued, you needed to manipulate just a few of the grid columns. You could use this system function to move all of the GC fields to GB without doing a copy for each individual column.
Delete Grid Row	Use this if you, not the runtime engine, are manipulating the grid row. For example if you are using the Delete Doc routine out of the master business function.
Disable Grid	Use this to disable the entire grid from input. This is useful if you need more information in the header filled in before individual rows can be added or to obtain information in the header to default into the grid.
Enable Grid	Enables the grid.

Get Grid Row	Use this to specifically read a grid row, for example, if you need to reprocess the grid through a While loop.
Get Max Grid Rows	Use this if you to determine how many rows must process if you need to reprocess all of the individual grid rows through a While loop. Use this just before the While loop.
Get Selected Grid Row Count	Use this to get the row number for a selected row. The only time you would probably need this is if you needed to save the row into a variable for future processing.
Hide Grid Column	Use this for dynamic processing.
Hide Grid Row	Use this to hide an entire row from display. For example, if you create a form with a grid and you want to summarize rows, you would add up several rows to determine a total and show the total row. If you use this system function, you could have a check box for the detail and then unhide the row instead of repopulating the grid.
Insert Grid Buffer Row	Use this to manipulate the buffer space. If you use this system function, GC becomes GB.
Insert Grid Row	Use this to insert a row into the grid. Usually these rows are custom (nondatabase), for example a totalling row.
Set Grid Cell Error	Similar to Set Edit Control Error.
Set Grid Color	Similar to Set Edit Control Color.
Set Grid Font	Similar to Set Edit Control Fonts.
Set Grid Row Bitmap	Use for manipulating the bitmap being displayed outside the grid. Sets a specific bitmap on a specified row header.
Set QBE Column Compare Style	Use for complex assignments to the QBE runtime structure. For example, if you want to assign the date January 1, 1998 to a particular QBE column, you use QC=010198. Because there are several valid comparisons for the QBE line, if you wanted the date to be $\geq$ January 1, you must use this system function.
Show Grid Column	Similar to Show Edit Control.

Suppress Grid Line	Use this 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	Once the grid buffer has been 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). An example of where this capability might be useful is in a customer service organization where customers call in for support. When a customer calls in, you can use your application to identify who is calling and automatically retrieve information about the caller.

You can use TAPI to do things such as:

- 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 that 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, you check to see if the user has requested a transfer and then you 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 OneWorld 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 has ended.

- End a call

Calls can be ended in several ways, including hanging up by the user or the caller, call transfers, or bad phone connections. You add event rules at the point that the application receives a message that the call has ended to properly close the call.

Refer to the Telephony System Functions under Online APIs for more information about specific system functions.

Attaching a Business Function to an Event

Use the business function button to attach an existing business function to an event. Business functions are either:

- Manually generated C code (source language C)
- OneWorld generated using Business Function Event Rule Design (source language NER)

Business functions are:

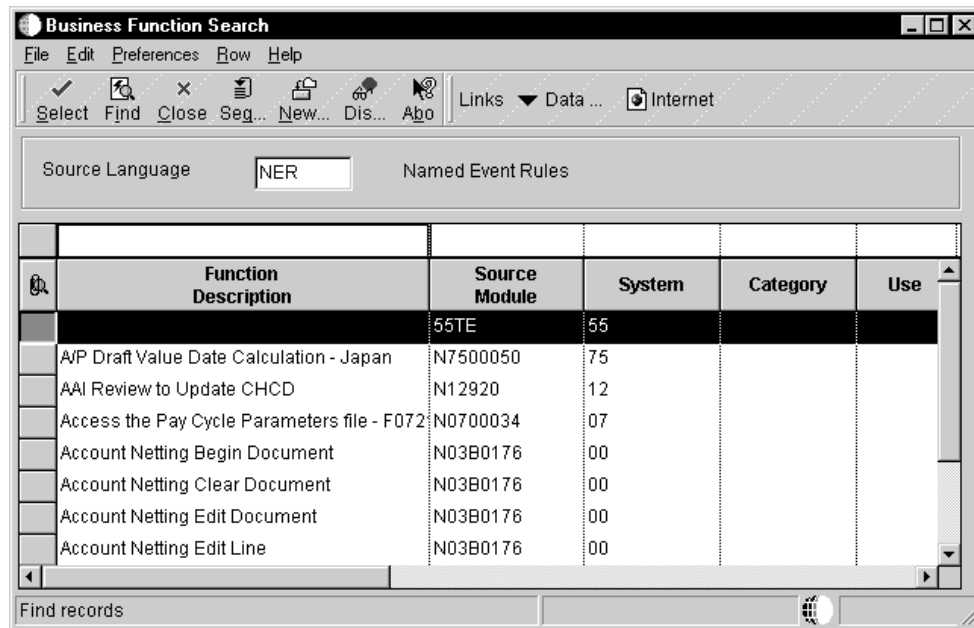
- Commonly used for referential integrity, such as deleting secondary records when a master record is deleted, and for editing routines
- Typically used for large and complex calculations that can otherwise overload the engine



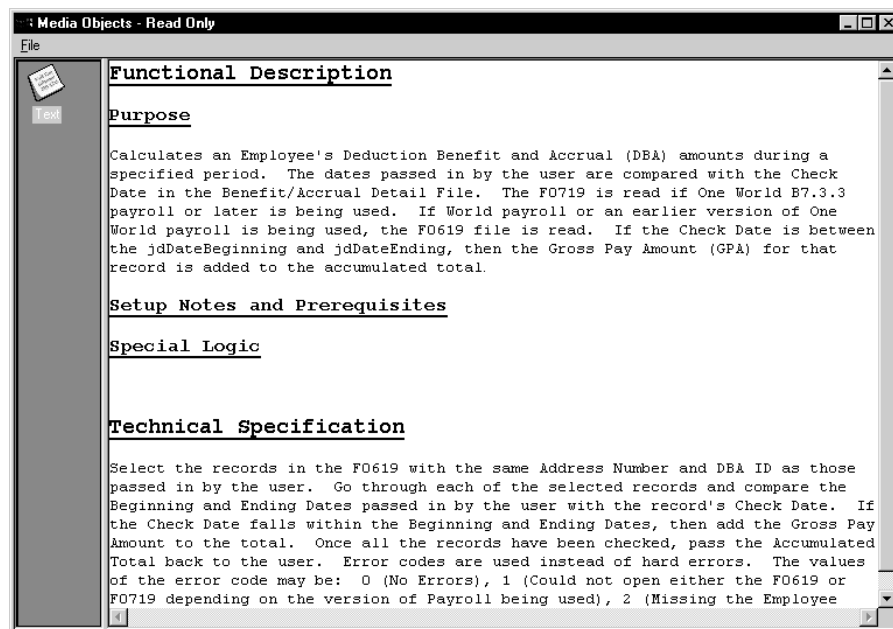
To attach a business function

On Event Rules Design

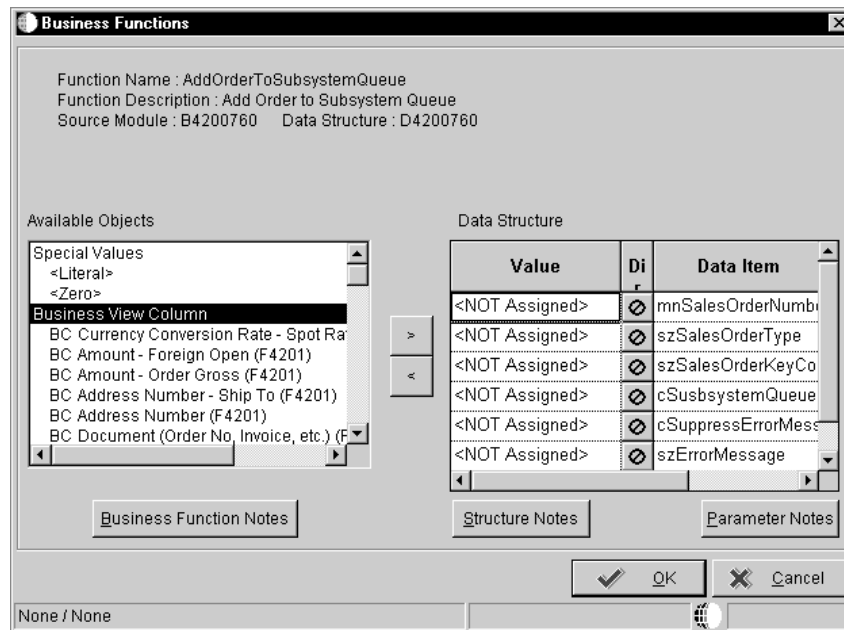
1. Choose an event.
2. Click the $f(B)$ button.



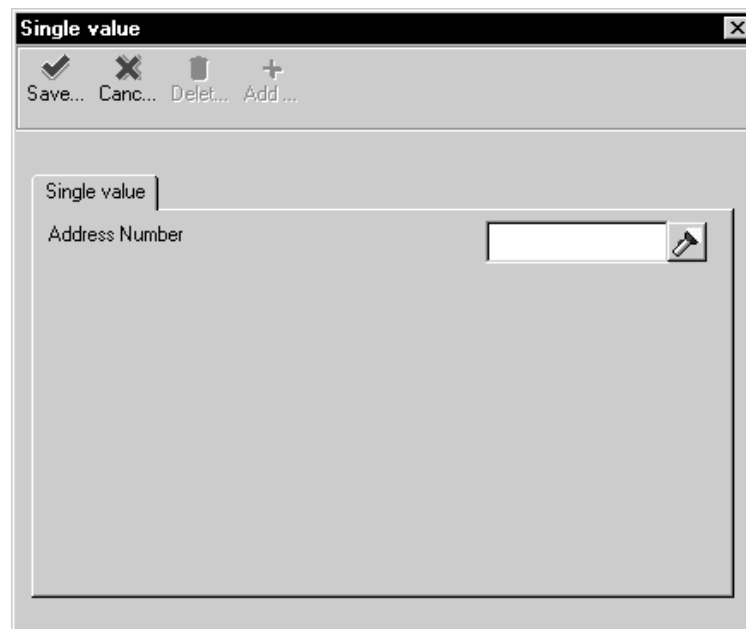
You can view a description for a business function by choosing Attachments from the Row menu.



3. Choose a business function and click OK.
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. Enter a single value and click OK.



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

6. 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

← 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 data structure for a business function has the direction of the items hard coded in the data structure (for example, if the parameters are predetermined to be input, output, or bidirectional), then this predetermined direction appears here. If they are optional they can be skipped or set by the user. Required items appear in red and must be completed. The status bar indicates the state of the flow to the target.

7. Turn on the Include in Transaction option to include this business function for transaction processing.

This option only appears if you are calling from a Fix/Inspect, Header Detail, or Headerless Detail form. Refer to *Transaction Processing* in this guide for more information about transaction processing.

8. Turn on 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. Refer to *Asynchronous Processing* in this guide for more information about asynchronous processing.

9. You can click one of the following buttons to add notes.

- Business Function Notes
- Structure Notes
- Parameter Notes

10. Click OK.

J.D. Edwards Design Standards

Always use the directional arrows to attach business functions. If a parameter is not used, then use the ⊘ symbol. This also:

- Identifies when a parameter has been forgotten or needs to be added, for example, if 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

Use numeric values for any event rule flags that are passed back from the business functions. This is more acceptable internationally. For example, to assign true/false values, use 1 for true and 0 for false, instead of T and F or Y and N.

Refer to the *Business Functions* section in this guide for more information about business functions.

Creating Form Interconnections

Use form interconnections to automatically pass data from one form, the source, to another form, the target.

This chapter describes the following:

- ☐ Creating a Modal Form Interconnection
- ☐ Creating a Modeless Form Interconnection

Modal Processing

Most form interconnections are modal. This means that after a form interconnects to another form, the user must close the second form before interacting with the first form again.

Modeless Processing

You can create a modeless form interconnection between a Find/Browse form and one or more Fix/Inspect forms or Transaction forms (Header and Headerless Detail forms). Modeless processing allows the user to interact with two or more forms at the same time instead of having to close one to return to the other. An update to the detail is reflected on the Find/Browse form without reinquiring on the database. After the initial Find/Browse, you can go to any form type, including another Find/Browse.

Each time a modeless form interconnection is called, the values in the form data structure are passed. When Cancel is clicked in the calling form, the called form is destroyed and no values are passed. When OK is clicked in the called form, values are passed back to the calling form through the form data structure. The called form still appears.

The *Dialog is Initialized* event only occurs the first time it is called. If you want event rules to run each time a form appears, you should attach them to *Post Dialog is Initialized*.

If a form in update mode fails to fetch a record from the database, the form mode is changed to Add mode. If you have the “Close Form On Add” option turned on, when you click OK the form will close.

If a Find/Browse form is closed, it goes through its list of modeless interconnect records and closes them before destroying itself.

Event rules following a modeless form interconnection are executed immediately instead of waiting for the called form to return.

Run applications with modeless processing from the menu. If you run an application from Object Librarian, modeless processing does not work.

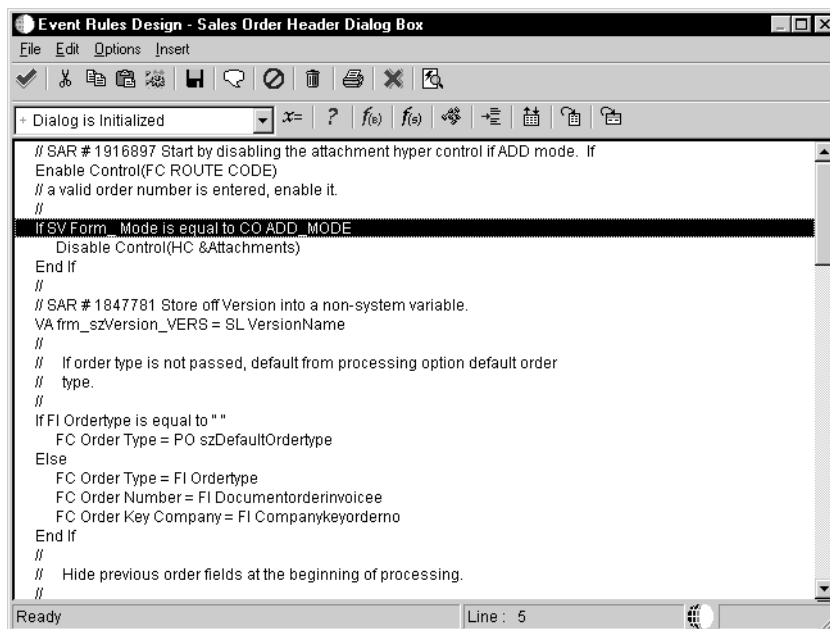
Before You Begin

Before you create a form interconnection, you must define the data structure of the receiving form to include any field for which values are being passed.

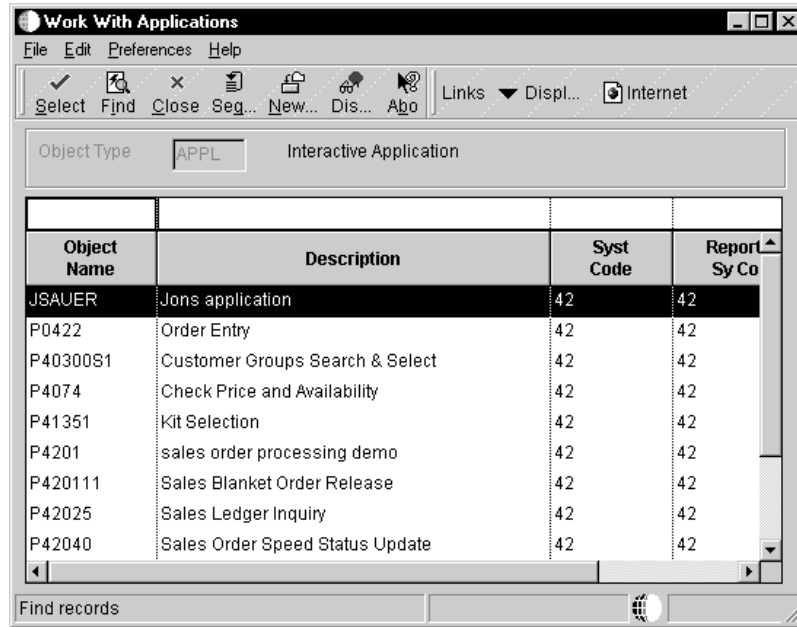
Creating a Modal Form Interconnection

► To create a modal form interconnection

1. On Event Rules Design, choose an event.

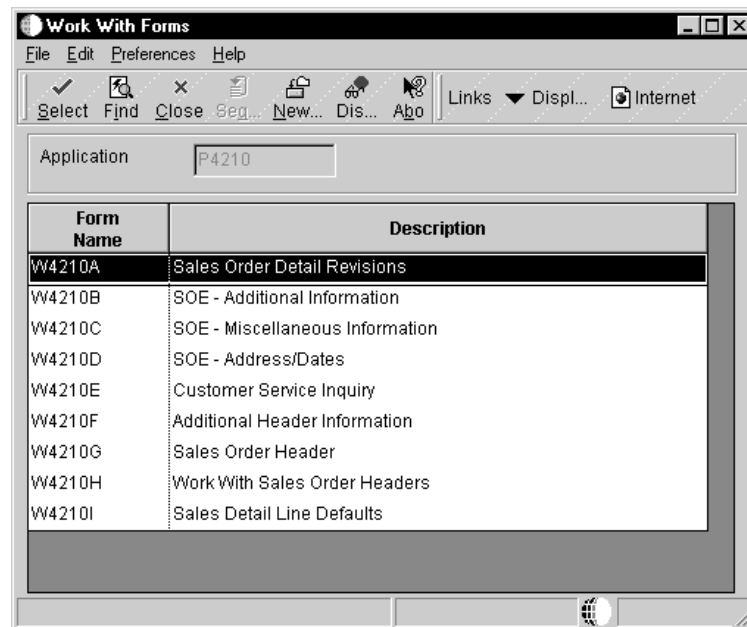


2. Click the Interconnect button.



- On Work with Applications choose the application to which you are connecting.

Work with Forms displays forms for the selected application.



- Choose the appropriate form to which you want to connect (the target).
- Choose the appropriate version of the form to which you want to connect.

Version	Version Title	Path Code	Prompt Options
ZJDE0007	Sales Order Entry - OT Order Type		2
ZJDE0008	Sales Order Entry - Configurator		1
ZJDE0009	Sales Quote -Configurator		1
ZJDE7001	Sales Order Entry - VO Order Type		2
ZJDE7002	Sales Order Entry - VO Order Type		2
ZJDE7003	Sales Order Entry - Return Order - Ship To		1
ZJDE7004	Sales Order Entry - Return Order - Ship From		1
ZJDE7005	Sales Order Entry - Reverse Order		1
ZJDE7006	Sales Order Entry - Return Order - Purchases		1

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.

Form Interconnect Name: Call(App:P4210, Form:W4210A) Version Name: ZJDE7001

Available Objects:

- Special Values
 - <Literal>
 - <Blank>
- Business View Column
 - BC Work Station ID (F4201)
 - BC Program ID (F4201)
 - BC User ID (F4201)
 - BC User Reserved Reference (F4201)
 - BC User Reserved Code (F4201)
 - BC Order Taken By (F4201)

Data Structure:

Value	Dir	Data Item
	⊗	DocumentOrderInvoic
	⊗	OrderType
	⊗	CompanyKeyOrderNo
	↔	Costcenter
BC Address Number -	→	Addressnumber
	⊗	Addressnumbershipit
	⊗	Datetransactionjulian

Structure Notes Parameter Notes

OK Cancel

- In the Available Objects column, select the object that you want to pass. 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 ⊗ 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.

Upon exiting the target, data flows back to the source

○ No data flows either way

8. Turn the Include in Transaction option to include this interconnection for transaction processing.

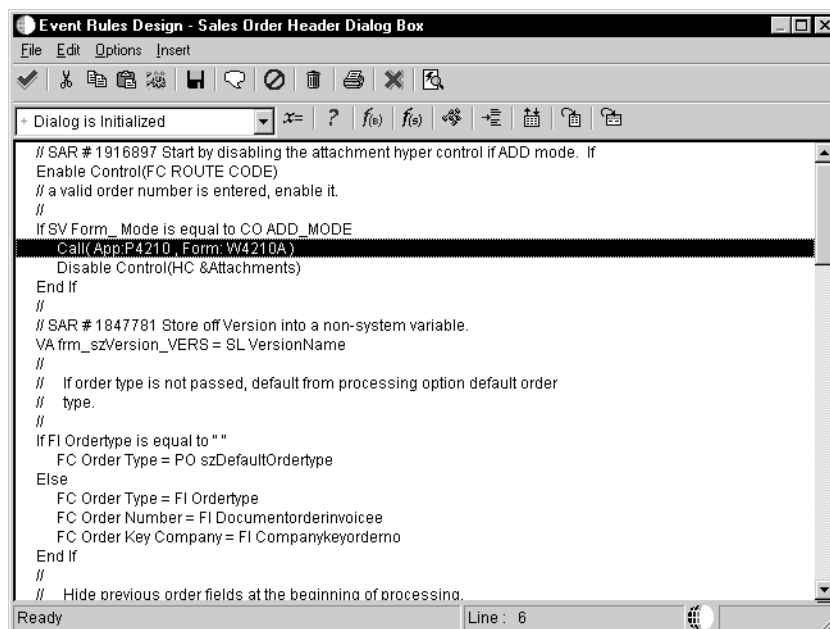
This option only appears if you are calling from a Fix/Inspect, Header Detail, or Headerless Detail form. Refer to *Transaction Processing* in this guide for more information about transaction processing.

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 statement: Call (Application <name> Form <name>).



J.D. Edwards Design Standards

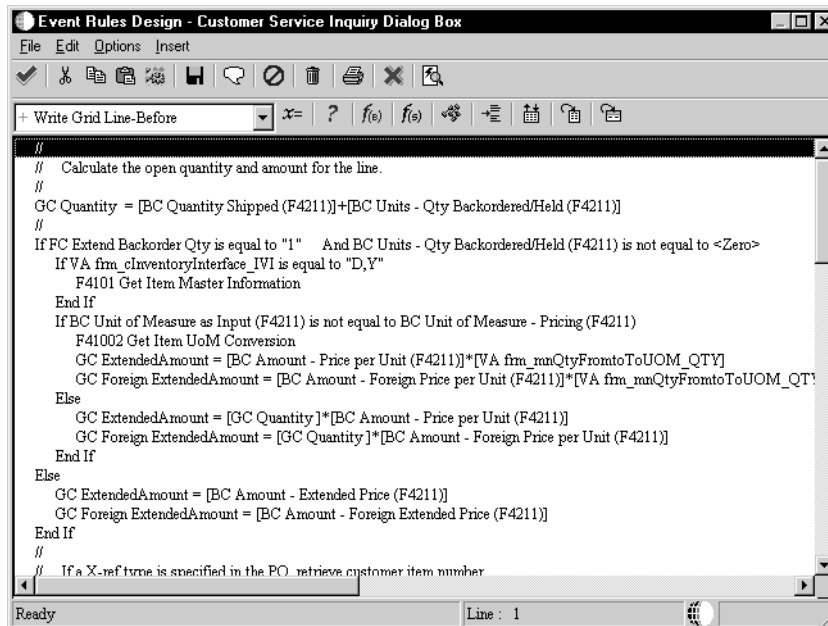
The following standards apply to all form types:

- 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.

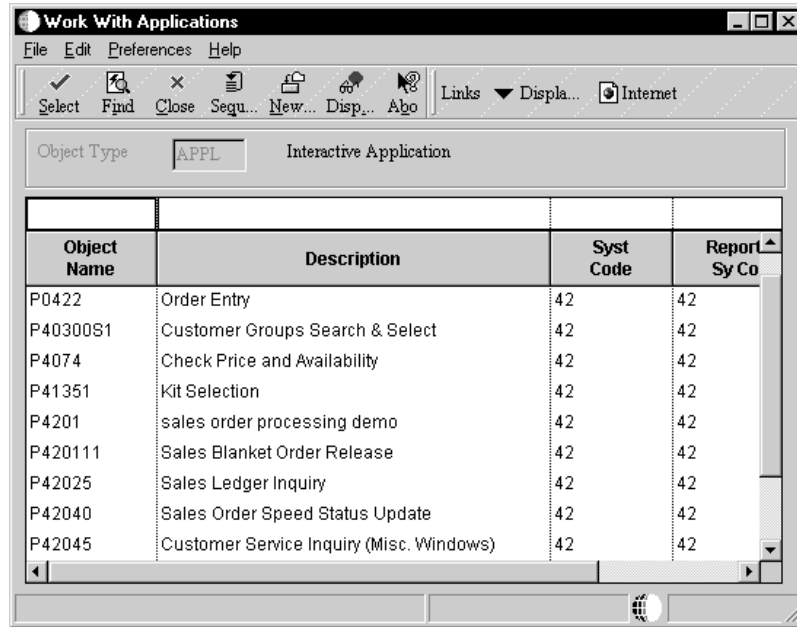
Creating a Modeless Form Interconnection

► To create a modeless form interconnection

1. On Event Rules Design for the Browse form you wish to use (the source), choose an event.

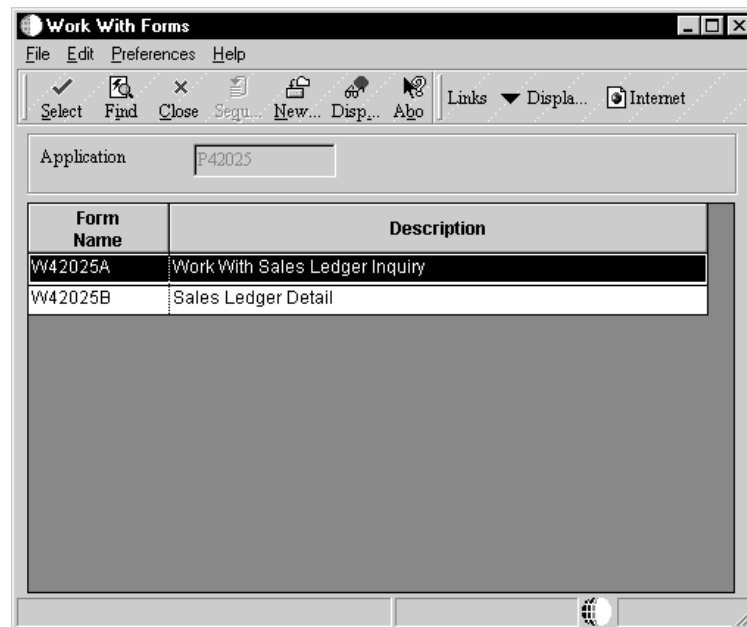


2. Click the Form Interconnection button.

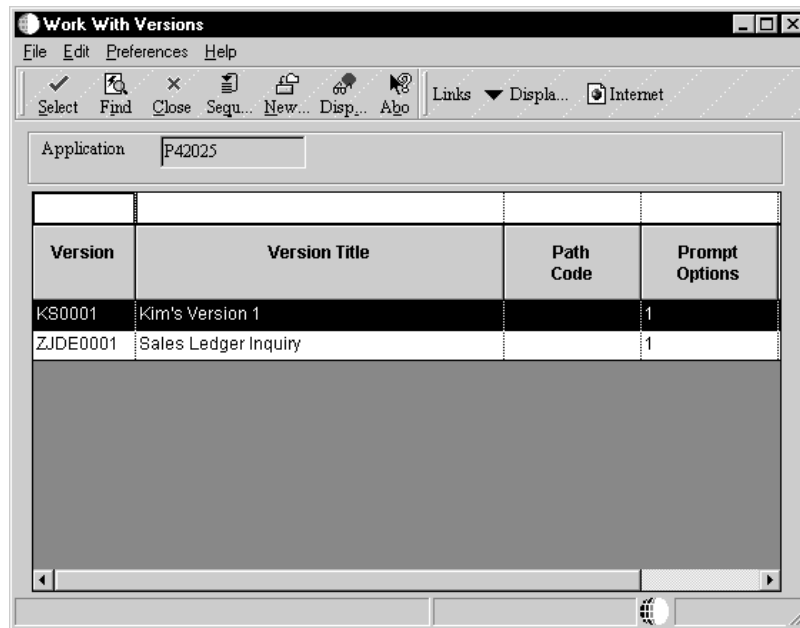


- On Work with Applications choose the application to which you are connecting.

Work with Forms displays forms for the selected application.

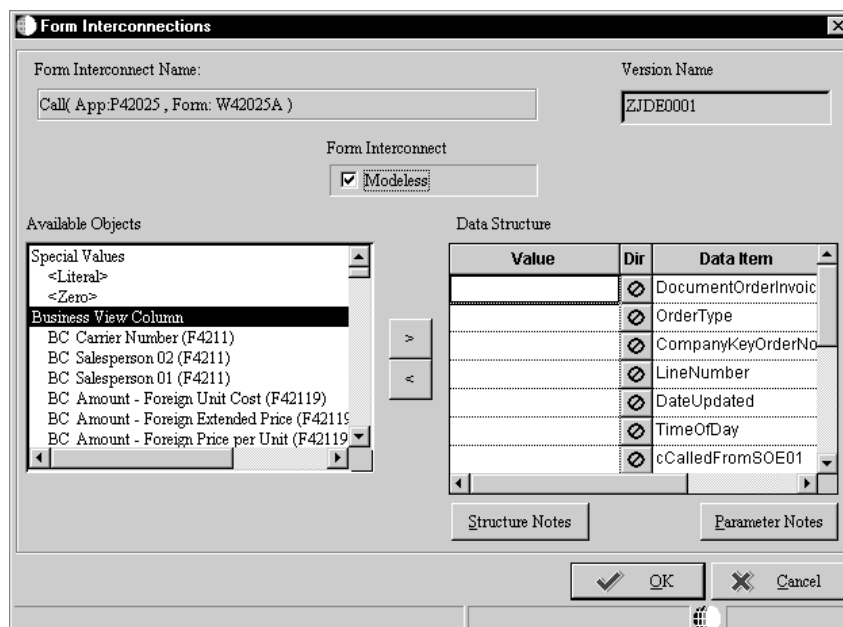


- Choose the appropriate Fix/Inspect form to which you want to connect (the target). The Form Interconnect - Values to Pass window displays the data structure for the target form.



- 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.



- Turn on the Modeless option.
- In the Available Objects column, choose objects that you want to pass. Use the > button to move it 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.

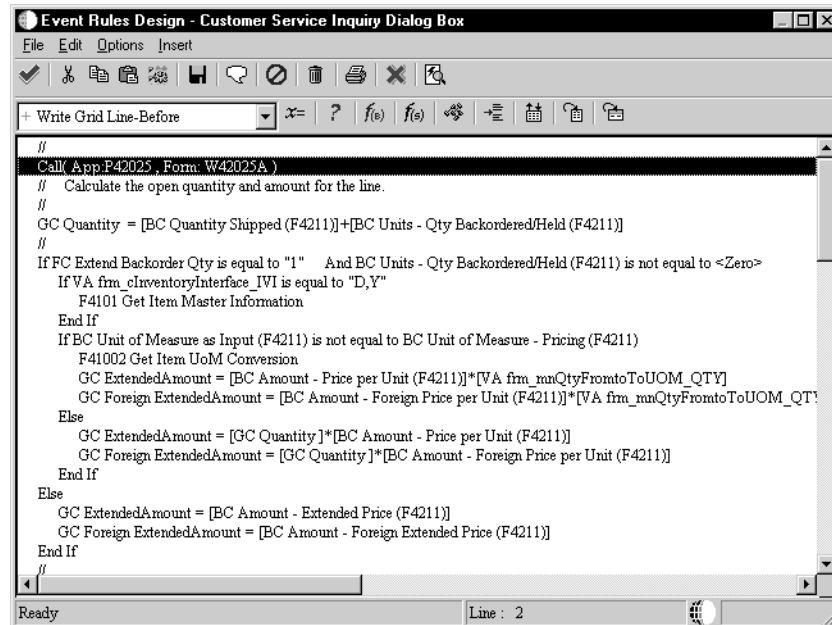
← Data flows from the target to the source.

↔ Data Flows from the source to the target, and upon exiting the target, data flows back to the source.

 No data flows either way

9. Click one of the following buttons to add notes:
 - Structure Notes
 - Parameter Notes
10. After the data structure is defined, click OK.

The Event Rules Design displays the Form Interconnection with the statement: Call (Application <name> Form <name>).



Creating Report Interconnections

Use report interconnections to automatically execute 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.

This chapter describes the following:

- ☐ Creating a report interconnection

Before You Begin

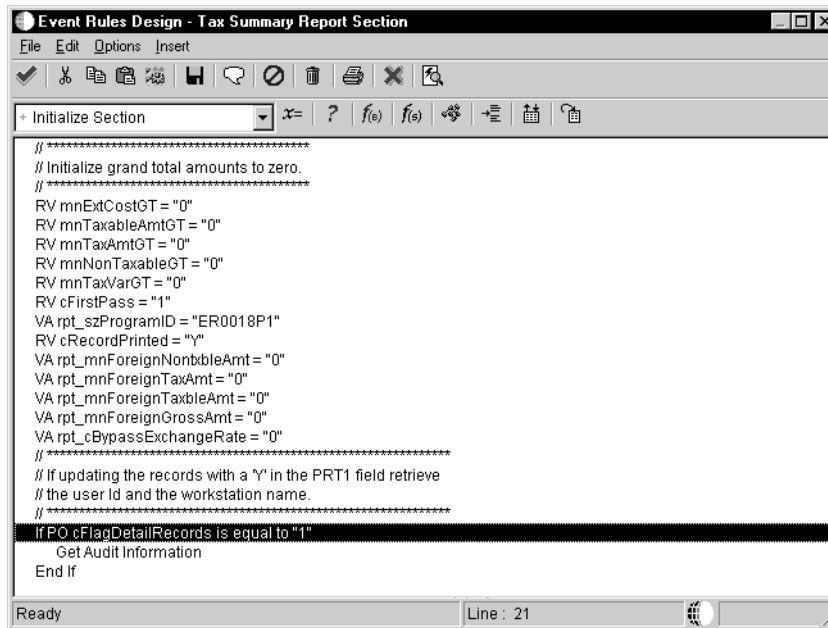
Before you create a report interconnection, you must define the data structure of the receiving form to include any field for which values are being passed.

Creating a Report Interconnection

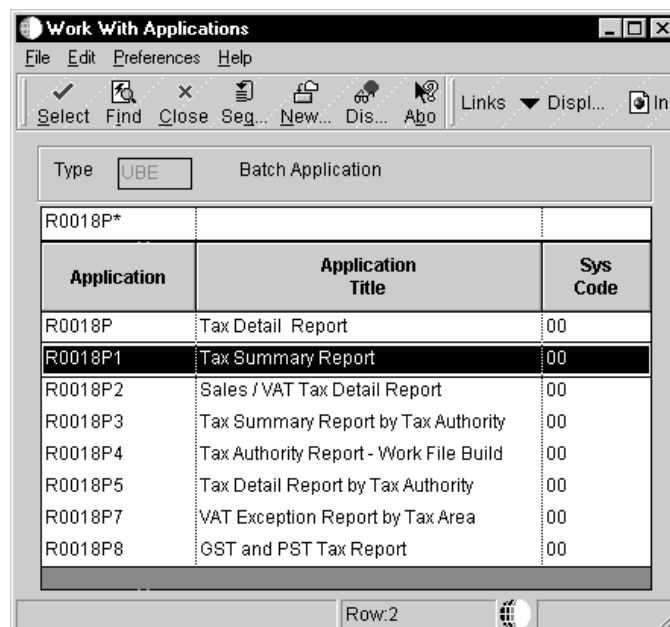


To create a report interconnection

1. On Event Rules Design, choose an event.

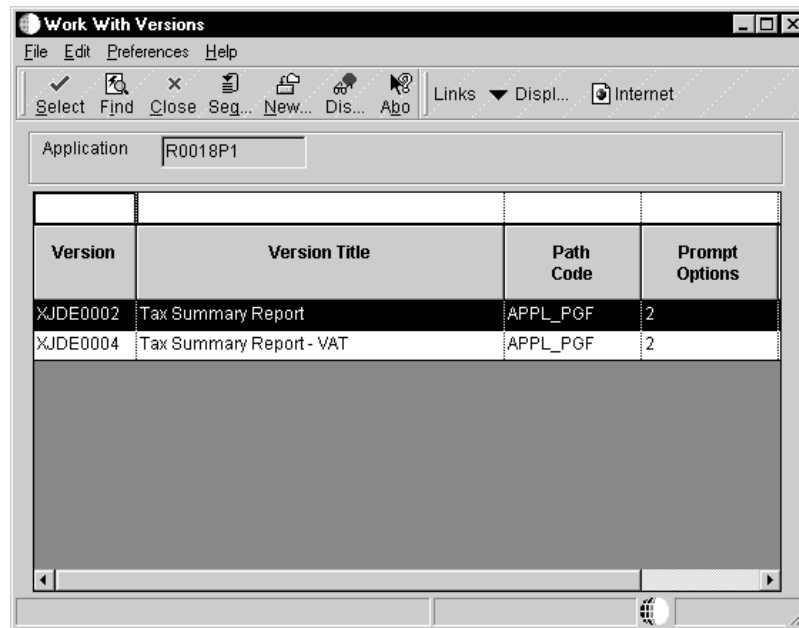


2. Click the Report Interconnection button.

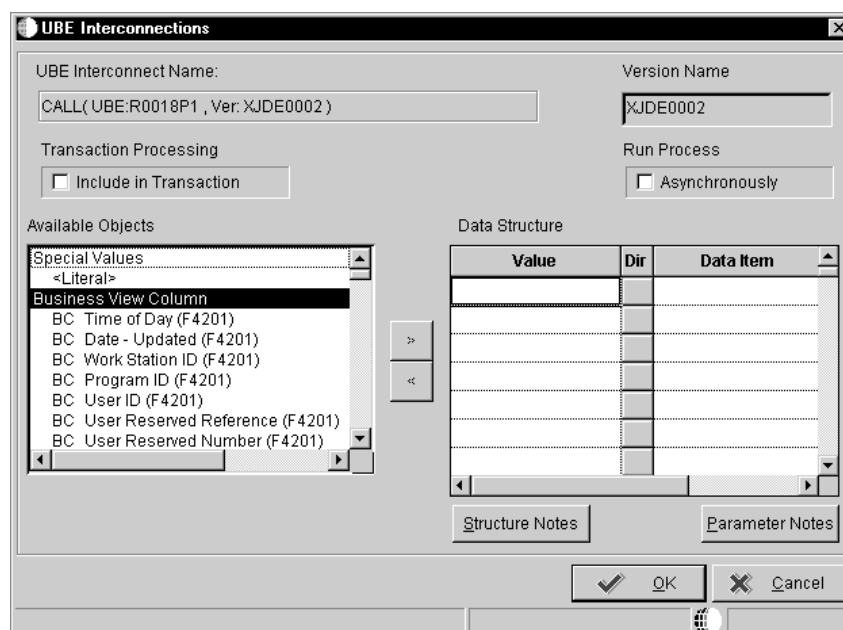


3. On Work with Applications choose the report to which you are connecting.

Work with Versions displays versions for the selected report.



- Choose the appropriate version of the report to which you want to connect.



- 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.
- 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 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.

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:

- Asynchronously

You 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

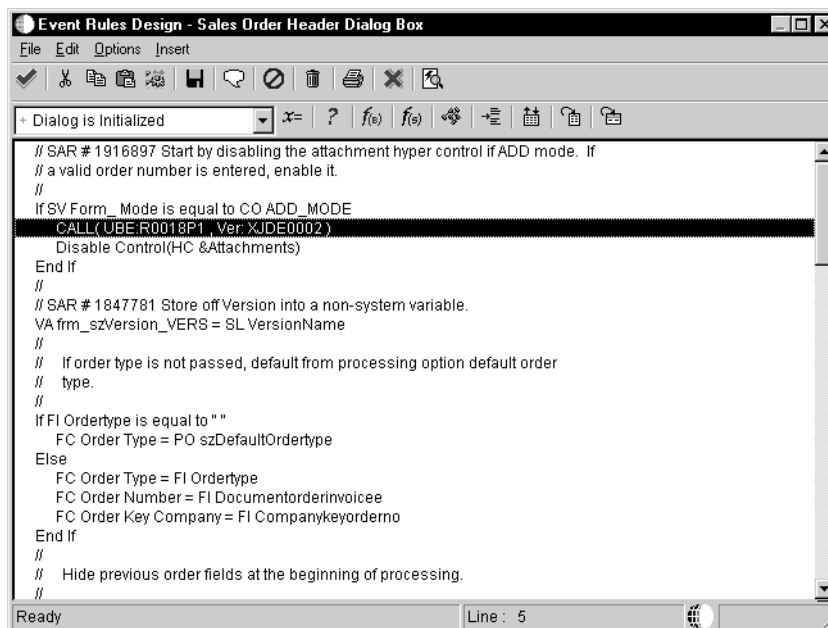
9. Click one of the following buttons to add notes:

- Structure Notes
- Parameter Notes

10. After the data structure is defined, click OK.

Event Rules Design displays the Report Interconnection with the 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 field is left blank, or calculates a value, rather than writing a business function to calculate it.

When you create an assignment you can:

- Assign a fixed value
- Create and assign a mathematical expression

When creating an expression, be careful to only sum data items that are of the exact same numerical scale or data type. For example, *do not* sum different currencies or decimal figures that represent different decimal values, as this would result in a loss of precision.

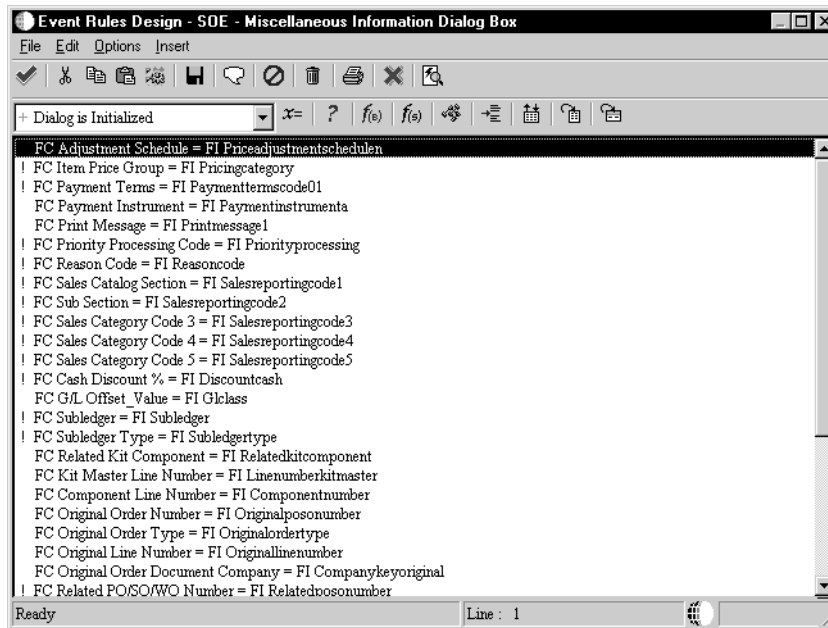
The available objects that appear are based on the data type.



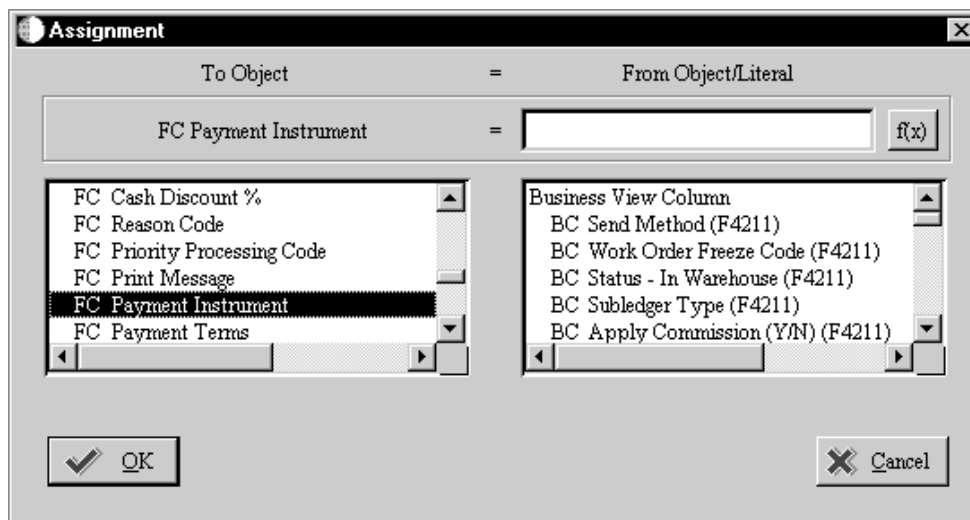
To assign a value

On Event Rules Design

1. Choose an event.



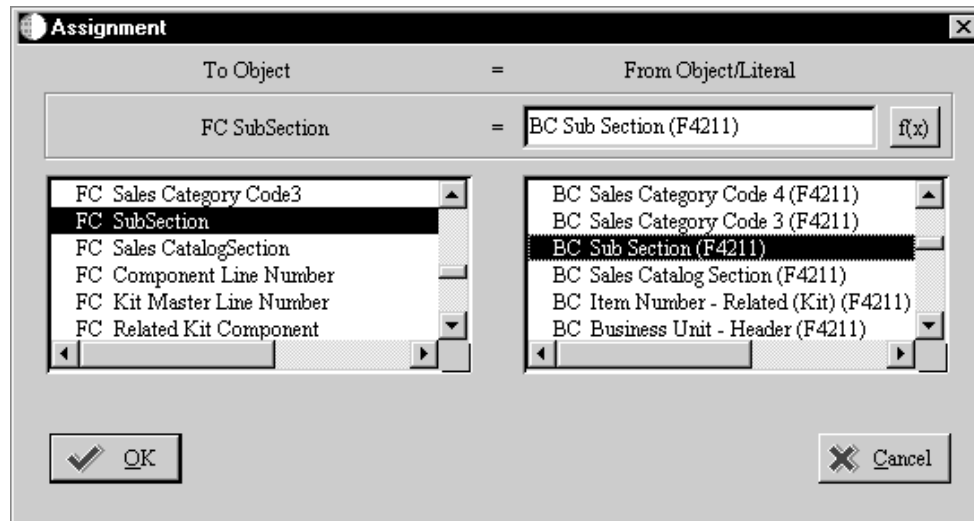
2. Click the Assign button.



3. Choose an object on the left that is to be the recipient for your assigned value.

4. Determine the “From” value using one of the following options:

- 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.



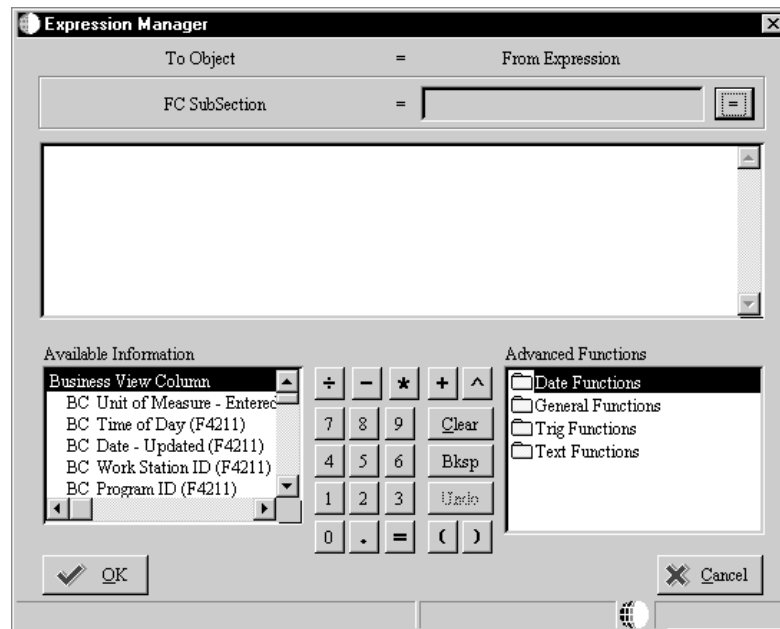
5. Click OK.

The Event Rules Design window displays the assignment in text form.

► To create an expression for an assignment

From the Assignment form

1. Click the $f(X)$ button.



2. Create the expression in the edit field. Either type the expression or use the object list, calculator pad, and functions list.

Use the following buttons as you 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 are finished editing your expression.

The Assignment Display checks and highlights any spelling or syntax errors. If no errors are detected, the expression is returned to Event Rules Design.

Expression Manager

Expression Manager has several different sections to work with.

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 being edited.

Expression Editing Field

Create and edit the expression in the editing field.

You can enter text by typing characters from the keyboard, using the editing keypad, or double-clicking the mouse to point and click to various expression components, including:

- Objects
- Numbers and mathematical symbols
- Advanced functions

The editing field will accept *any expression, in any order*. If syntax errors occur, the system will attempt to highlight and display the error in the status bar.

Available Information

The Available Information list contains all the objects available for creating and editing expressions. You can choose an object in the list box to display 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 only enabled when it can undo the last editing operation performed. It cannot undo inserts from either the Available Information list or the Advanced Functions list.

Advanced Functions

The Advanced Functions list contains functions 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
- Trig Functions
- Text Functions

Date Functions

<u>Function</u>	<u>Description</u>
add_days(date, days)	Adds specified number of days to the date
add_months(date, months)	Adds 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 today's 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 week)
next_day(date, day_of_week)	Returns the next date that falls on the day of the week after a given date

General Functions

<u>Function</u>	<u>Description</u>
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 sign is negative returns -1, else returns 0)
sqrt	Calculates the positive square root of the specified number

Trig Functions

<u>Function</u>	<u>Description</u>
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

<u>Function</u>	<u>Description</u>
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 <code>lower("AbCdEfG") = "abcdefg"</code>
lpad	Inserts the character into the front (left side) of the string until the string is length long <code>lpad("qwerty", ' ', 12) = "..... qwerty"</code>
ltrim	Removes leading occurrences of the character from the string <code>ltrim("aaaaaabcdef" , 'a') = "bcdef"</code>
rpadd	Inserts the character into the back (right side) of the string until the string is length long <code>rpadd("qwerty" , ' ', 10) = "qwerty...."</code>
rtrim	Removes trailing occurrences of the character from the string <code>rtrim("abcdef" , ' ') = "abcdef"</code>
substring	Extracts a portion of string beginning at position for length number of characters <code>substring("SUNMONTUEWEDTHUFRIS AT", 9,3) = "WED"</code> 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.
upper	Converts specified string to uppercase <code>upper("qweRTY") = "QWERTY"</code>

Table I/O

Use the Table I/O button in Event Rules Design to create instructions that perform table input and output (I/O), without having 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:

- Validate data
- Retrieve records
- Update or delete records across files
- Add records

You can use Log Viewer to view your table I/O SQL statements in the `jddebug.log`. (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. Opens a table for I/O but does not close it. All tables that do not use handles are automatically closed when the form using them is closed.
Insert	Inserts a new row.
Update	Updates an existing row. Only those columns mapped (presently in tables with or without handles) will be updated. It is possible to do partial key updates with tables and handles to tables. If you do not specify all the keys then several records may be updated.
Delete	Deletes a row(s) in a table or business view.
Open	Opens a table or business view.

Close	Closes a table or business view.
Select	Selects row(s) for a subsequent FetchNext operation.
SelectAll	Selects all rows for a subsequent fetch next operation.
FetchNext	Fetches rows that have been selected. Multiple records can be fetched 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:

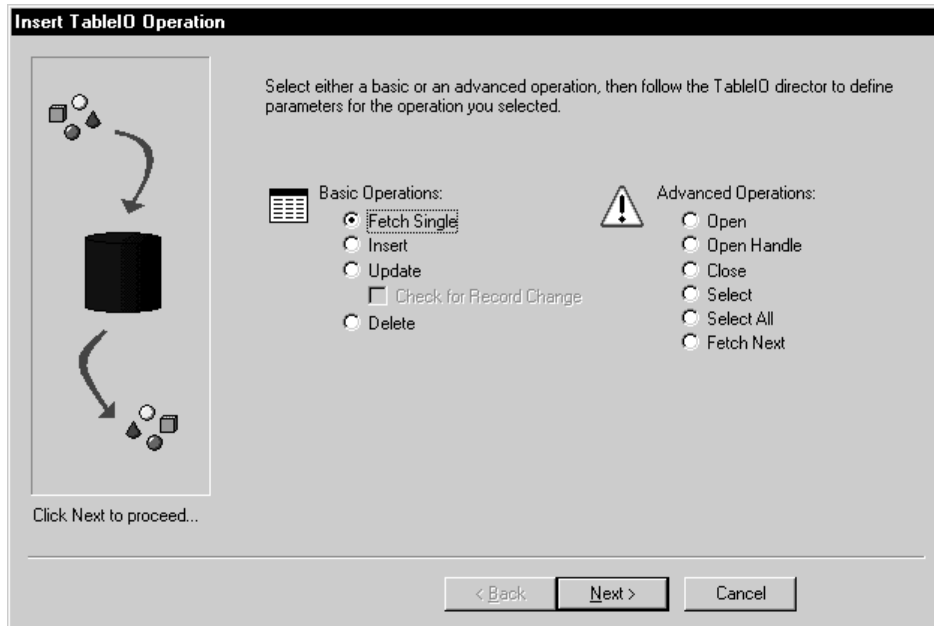
FetchSingle	Index Fields: =, <, <=, >, >=, !=, Like Non-Index Fields: Copy Target
Insert	All Fields: Copy Source
Update	Index Fields: = Non-Index Fields: Copy Source
Delete	All Fields: =
Open	N/A
Close	N/A
Select	All Fields: =, <, <=, >, >=, !=, Like
SelectAll	N/A

Creating a Table I/O Event Rule

► **To create a table I/O event rule**

On Event Rules Design

1. Click the Table I/O button.



Insert TableI/O Operation

Select either a basic or an advanced operation, then follow the TableI/O director to define parameters for the operation you selected.

Click Next to proceed...

Basic Operations:

- ☒ Fetch Single
- ☐ Insert
- ☐ Update
- ☐ Check for Record Change
- ☐ Delete

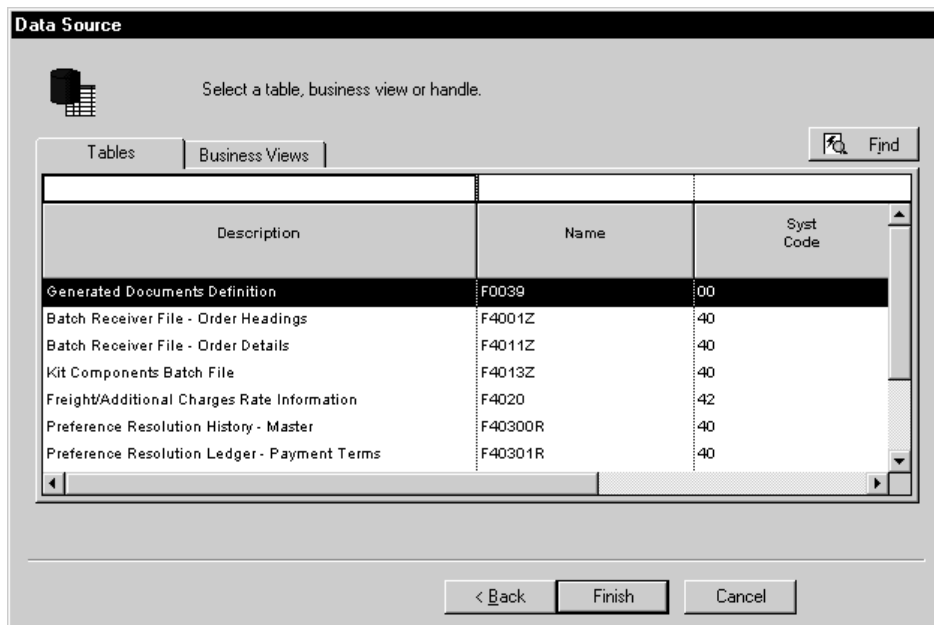
Advanced Operations:

- ☐ Open
- ☐ Open Handle
- ☐ Close
- ☐ Select
- ☐ Select All
- ☐ Fetch Next

< Back Next > Cancel

2. Choose the operation you wish to use and click Next.

If you choose the Open or Open Handle operation the handle option will not appear on the Data Source form. The Handle tab appears for all other options.



Data Source

Select a table, business view or handle.

Tables Business Views Find

Description	Name	Syst Code
Generated Documents Definition	F0039	00
Batch Receiver File - Order Headings	F4001Z	40
Batch Receiver File - Order Details	F4011Z	40
Kit Components Batch File	F4013Z	40
Freight/Additional Charges Rate Information	F4020	42
Preference Resolution History - Master	F40300R	40
Preference Resolution Ledger - Payment Terms	F40301R	40

< Back Finish Cancel

3. On Data Source, choose the table or business view for which you want to perform I/O.
4. 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. On FETCH statements, it maps values back 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.

The Mapping dialog box is titled "Mapping" and contains the instruction "Select an index and define mappings for this operation." It features a "Daylight Savings Rules" section with a text field containing "F00085.FetchSingle". Below this is a list of "Available Objects" including Business View Column, Form Control, Processing Option, Form Interconnect, System Value, and Variable. To the right, the "Index:" section shows a dropdown menu with "Rule Name" selected. The "Mapping:" section contains a table with three columns: "Column Name", "Oper Code", and "Variable Name". The table lists several columns from "DaylightSavingsRuleName" to "DsavTimeStart", each with a corresponding operator icon. At the bottom are buttons for "< Back", "Finish", and "Cancel".

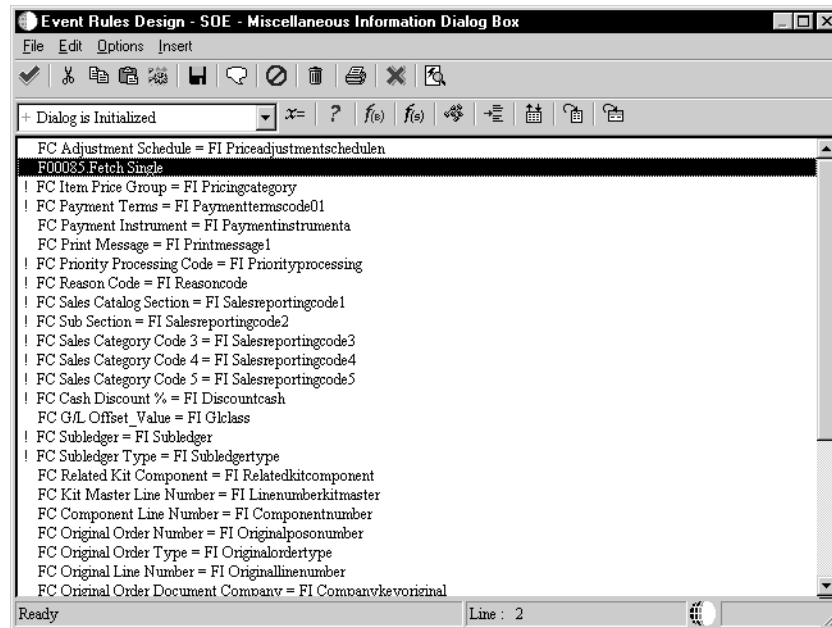
Column Name	Oper Code	Variable Name
* DaylightSavingsRuleName	⊗	
Description	⊗	
DaylightSavingsRuleType	⊗	
DsavInstanceStart	⊗	
DsavDayOfWeekStart	⊗	
DsavMonthStart	⊗	
DsavDayStart	⊗	
DsavTimeStart	⊗	

5. Choose the column you wish to use and double-click on the available objects you wish to map to that column.
6. Click the Operator button until you have the operation you want.

The following selection operations are available. The default is equal.

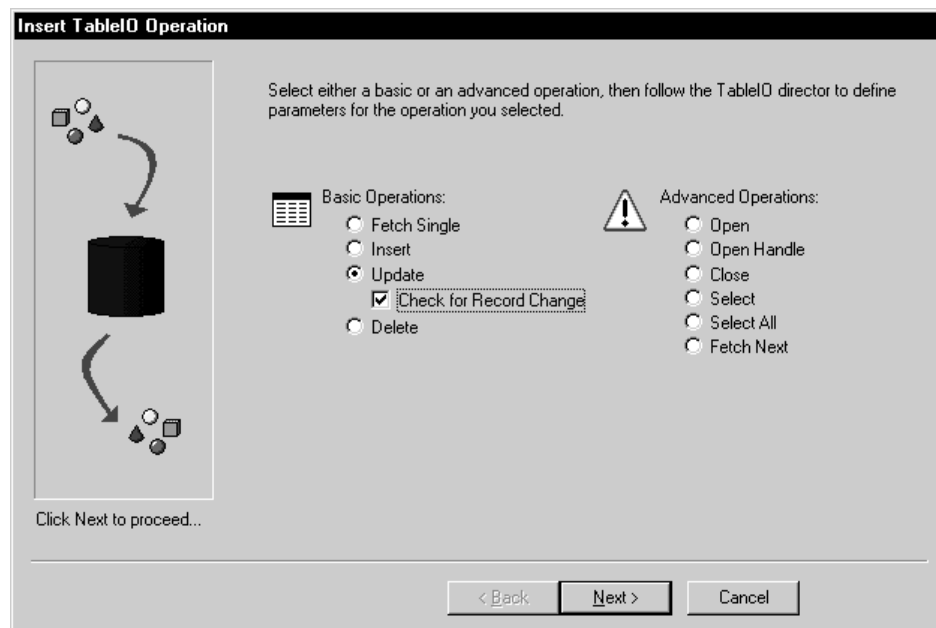
- Equal
- Not Equal
- Less Than
- Less Than or Equal To
- Greater Than
- Greater Than or Equal
- Like

7. Click Finish to save the operation and return to Event Rules Design.



Record Locking

If you choose the Update option, you can also enable Check for Record Change. Check for Record Change will tell you if the record has been changed between the time that it was fetched and the time that it is updated.



Understanding Handles

Handles allow you to do several things that you cannot do using nonhandle table I/O operations.

- You can use handles to concurrently open multiple instances of a single table or business view.
- You can use handles to open a table or business view in an environment other than the environment you are logged into.
- You can pass handles into a form or named event rule so that you do not need to open a table or business view more than once.

Handles cannot be used in transaction processing.

You must explicitly open and close handles, unlike other table I/O operations where you can use implicit open and close statements. You must open a handle before it can be used 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 the same way that you would close a table or business view except that you choose a handle instead of a table or business view.

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 just like any other handle.

Creating a Handle

There are several steps you must complete to use a handle. You should

- Define the handle in the data dictionary
- Create a handle variable in event rules
- Explicitly open the handle

To use handles you must define the handle in the data dictionary. You use a Class Type of Handle. You must use a Data Type of 7. The Size must be 11.

Work With Data Dictionary Items - [Data Item Specifications]

File Edit Preferences Form Window Help

OK Cancel Glossary Display Abort Links Glossary Internet

Data Item: HandleF0010 Alias: HF0010

Glossary Group: Primary Data Elements Item Parent:

General Information

Description: Handle System Code: 00

Data Type: 7 Identifier (ID): 11 File Decimals: 0

Class: HANDLE Display Decimals: 0

Control Type: 4 Generic Edit Item Occurrences: 1

Row Description: Handle

Column Title:

High Level Default Triggers

Default Value: ☐ Next Number: ☐

Visual Assist: ☐

Edit Rule: ☒ HNDL.F0010 ☐ Upper Case Only ☐ Row Security

Display Rule: ☐ ☐ Allow Blank Entry ☐ Auto Include

Once you choose the Handle data type, the Edit Rule Default Trigger option changes to Handle.

Work With Data Dictionary Items - [Data Item Specifications]

File Edit Preferences Form Window Help

OK Cancel Glossary Display Abort Links Glossary Internet

Data Item: HandleF0101A Alias: F0101A

Glossary Group: Primary Data Elements Item Parent:

General Information

Description: Handle for the table F0101 System Code: H94

Data Type: 7 Identifier (ID): 11 File Decimals: 0

Class: HANDLE Display Decimals: 0

Control Type: 4 Generic Edit Item Occurrences: 1

Row Description: Handle for the table F0101

Column Title: Table Handle

High Level Default Triggers

Default Value: ☐ Next Number: ☐

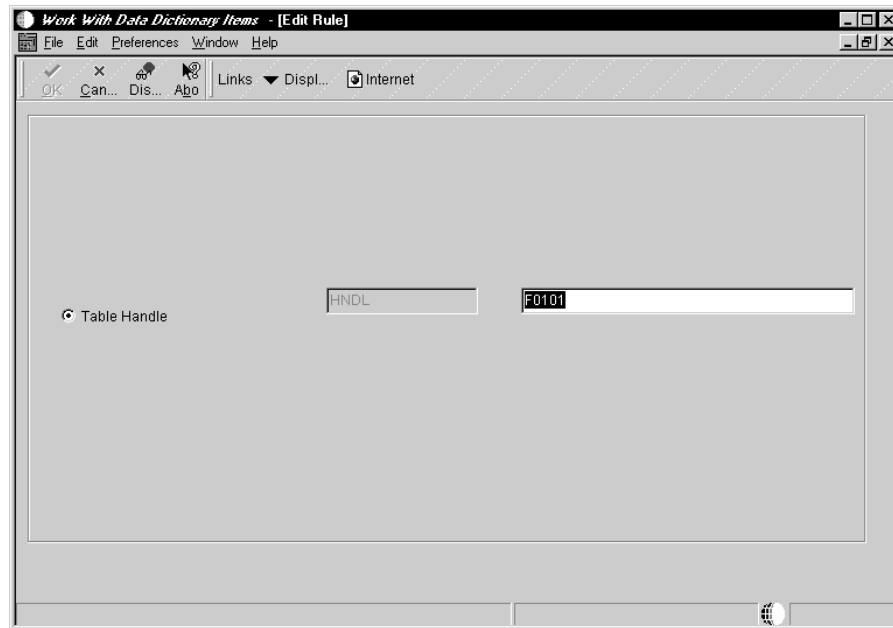
Visual Assist: ☐

Edit Rule: ☒ HNDL.F0101 ☐ Upper Case Only ☐ Row Security

Display Rule: ☐ ☐ Allow Blank Entry ☐ Auto Include

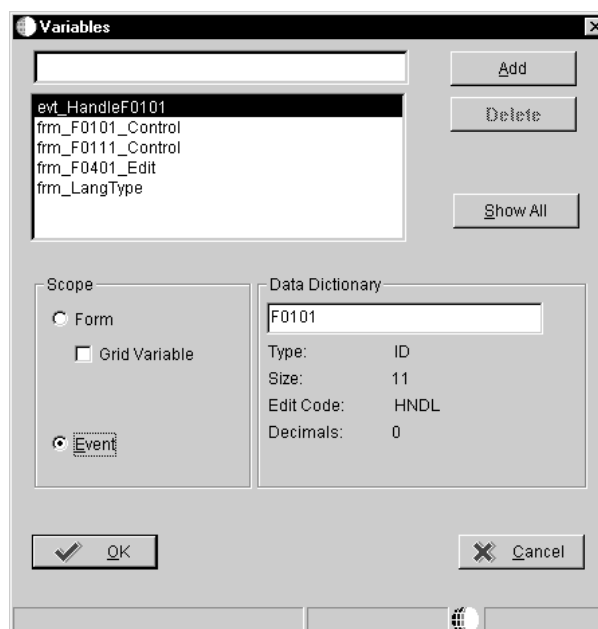
Save record

To complete the handle data item, click the Handle button and type in the table or business view name for the handle.



Refer to the Data Dictionary section in this guide for more information about data items. You can also create a handle data item in a data structure.

Once you create a handle data item, you must create a handle variable. You create a handle variables just like other variables. You can use any scope necessary to create the handle variable. Refer to *Working with Event Rule Variables* in this guide for more information about creating variables.



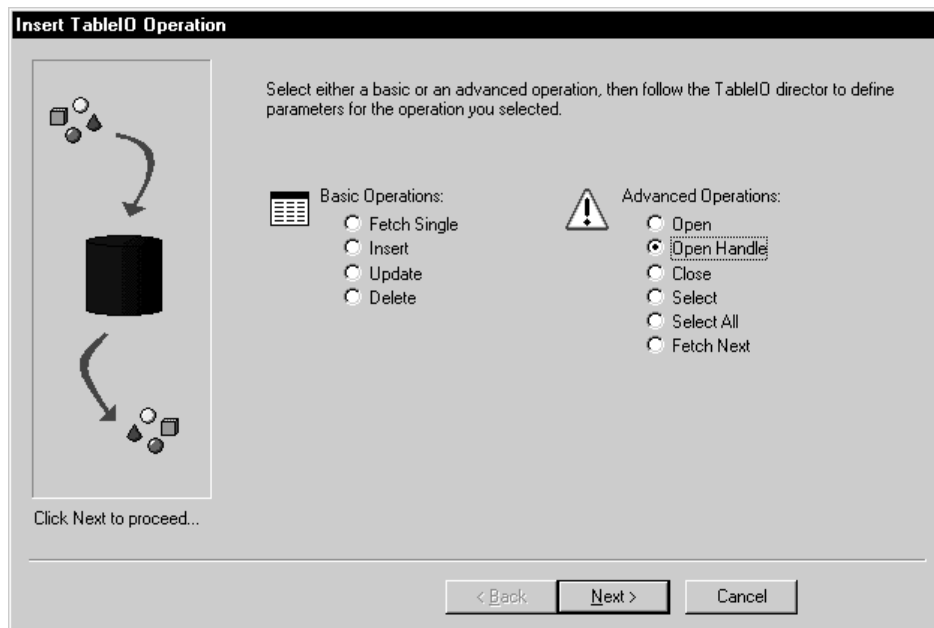
Opening Handles

After you create a handle variable, you must explicitly open the handle.

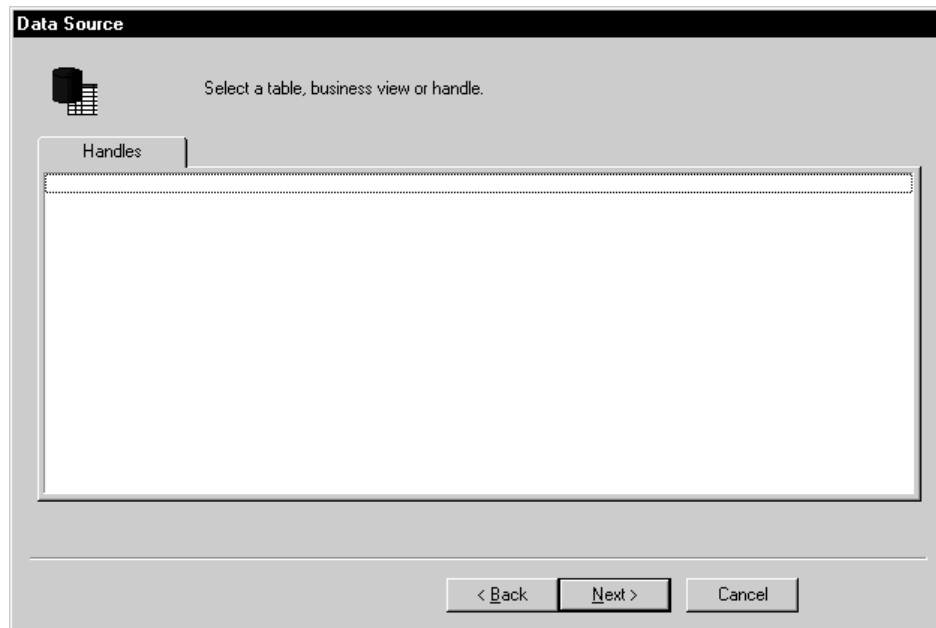
► To open a handle

On Event Rules Design

1. Click the Table I/O button.



2. From the Advanced Operations options, choose Open Handle.



3. Choose the handle you wish to open.

You should use the Handle class to create your handle data items. You should name the alias for the handle the same as its table.

4. 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 LoginEnvironment. System values also exist for the source and target environment in Table Conversion if you use Table I/O in Table Conversion.

5. Click Finish.

Table Event Rules

You use table event rules (TER) 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 a OneWorld database trigger, you must first determine which event will activate the OneWorld database trigger. Then 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 will be notified of changes or events to the table. There is no form or report interconnection available with table event rules.

You can use table event rules for data integrity, for example, when you delete a record in address book and you 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 is used to handle currency information in table event rules. Refer to Currency in this guide for more information on currency.

Following are 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

Refer to the online help Published APIs for information, such as typical use and processing sequence, about these events.

Creating Table Event Rules

To create a table event rule, you must perform the following:

- Create the database trigger in Event Rules Design

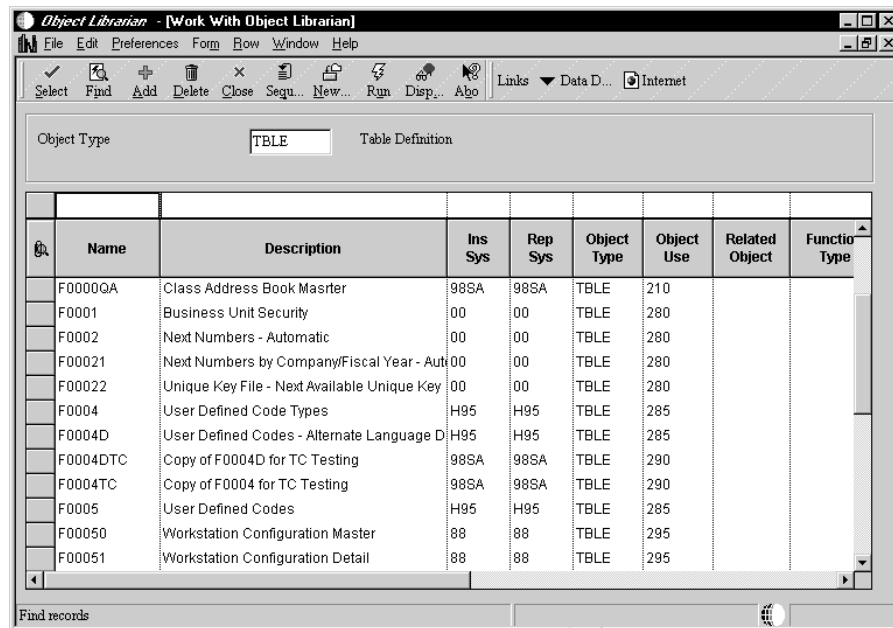
- Generate the OneWorld database trigger as C code

► To create table event rules

On Object Librarian

1. Choose the table to which you want to attach event rules.

You must check the table out to your local machine to attach event rules to the table.



2. On Table Design Aid, choose Table ER from the Form menu.

This starts Event Rules Design, which you can use to attach event rules to any of the events for the table.

Object Librarian - [Table Design Aid]

File Edit Preferences Form Row Window Help

OK Cancel New... Design Bro... Bul... Gene... Chec... Chec... Eras... Disp... Abo

Links Design Internet

Object Name: F42003 Object Type: TABLE

Description: Related Salesperson File

System Code: 42 Sales Management

Reporting System Code: 42 Sales Management

Object Use: 280 Constants Files Column Prefix: RE

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1 1		TN5669897	1/1/96	00:00:00	No change
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3 1		BS5625464	8/4/98	10:10:24	No change

Row: 1

- From the Events list, choose an event.

Event Rules Design - Related Salesperson File Table

File Edit Options Insert

Before Record is Deleted

Ready

- Click one of the following event rule buttons:

Business Function Attaches an existing business function

System Function Attaches an existing J.D. Edwards system function

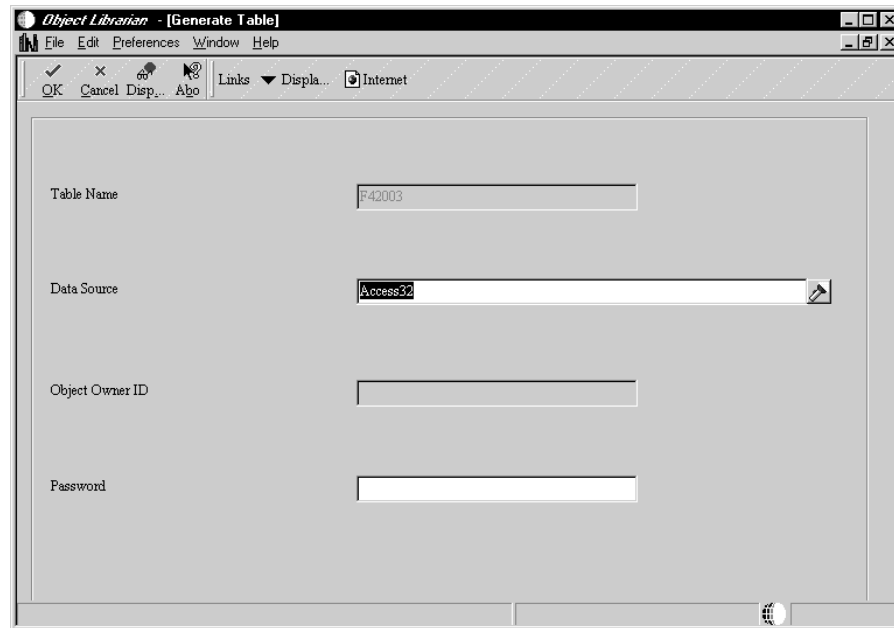
If/While	Creates an IF/WHILE conditional statement
Assign	Creates an assignment or a complex expression
Else	Inserts an ELSE clause, which is only valid within the bounds of IF and ENDIF
Variables	Creates a programmer-defined field, which has data dictionary characteristics for application-specific purposes, 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 (TER) functions. The table itself is the data structure that gets passed to the table event rule function.

5. On Event Rules Design, click Save to save your event rule specifications, then click Close to return to Table Design.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1	TN5669897	1/1/96	00:00:00	No change
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3	1	BS5625464	8/4/98	10:10:24	No change

6. On Table Design Aid, choose Generate Table from the Form menu.



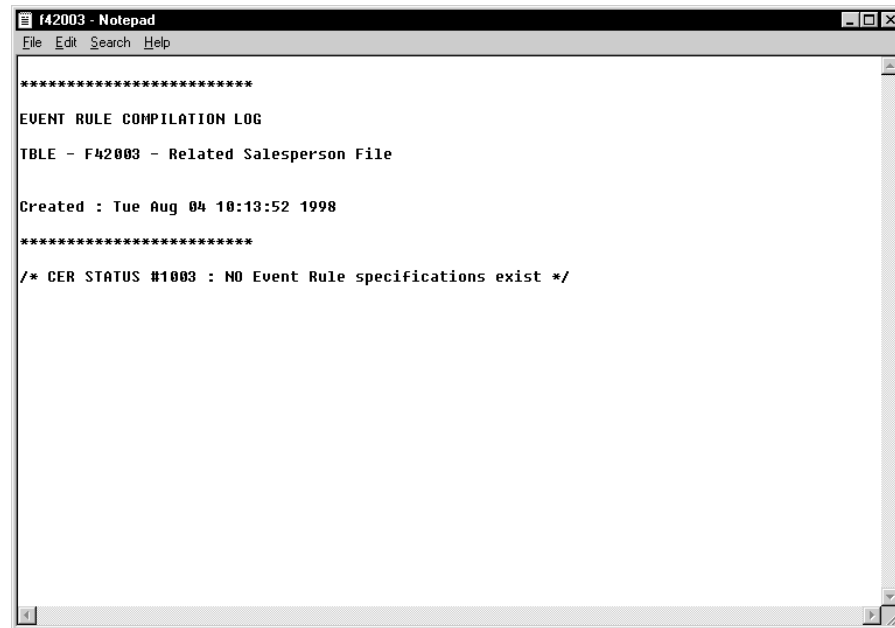
7. On Table Design Aid, choose Build Triggers from the Form menu.

The Build Triggers option performs the following steps:

- Converts the ER to C source code. This 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 is the Consolidated DLL that contains table event rule functions.

8. Click yes to display the ER status log.

The log will display the status of the build.



```
f42003 - Notepad
File Edit Search Help

*****
EVENT RULE COMPILATION LOG
TBLE - F42003 - Related Salesperson File

Created : Tue Aug 04 10:13:52 1998

*****

/* CER STATUS #1003 : NO Event Rule specifications exist */
```

The creation of the table event rule is complete. The newly created or modified table event rule functions will 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 where the control properties (such as row or grid description, visual assist, and edit rules) are not known until the users enters specific information or specific data is loaded. In these cases, the control properties must be dynamically overridden at runtime. For example, an application may need to display the ending date for a field instead of static text or you may want to change the visual assist search form when a user click 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 override form controls and grid columns that are data dictionary items
- To change the data dictionary item completely so that all data dictionary properties are changed
- To 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 all form controls and grid columns
- To change a specific data dictionary property
- If the data dictionary item is 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 use 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. When a thread is created, it is like calling a function except that the main program also continues running. This means that there are two points of execution progressing through the code. The different threads share the same memory space and are assigned processor time by the operating system.

Threads allow you to process selected OneWorld events or business functions in the background while the user continues to interact with the application.

Thread processing provides several performance benefits, including:

- Improved grid and edit control processing
- Background processing for certain events so the user does not have to wait for the system to finish
- Greater flexibility for event processing
- Multiple task processing
- Allows the user to continue with interactive tasks, such as data entry
- Allows faster cursor movement

Though threads can significantly improve performance by utilizing idle time to accomplish background processing, they also introduce a number of possible problems that 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 are trying to access the same data at the same time.

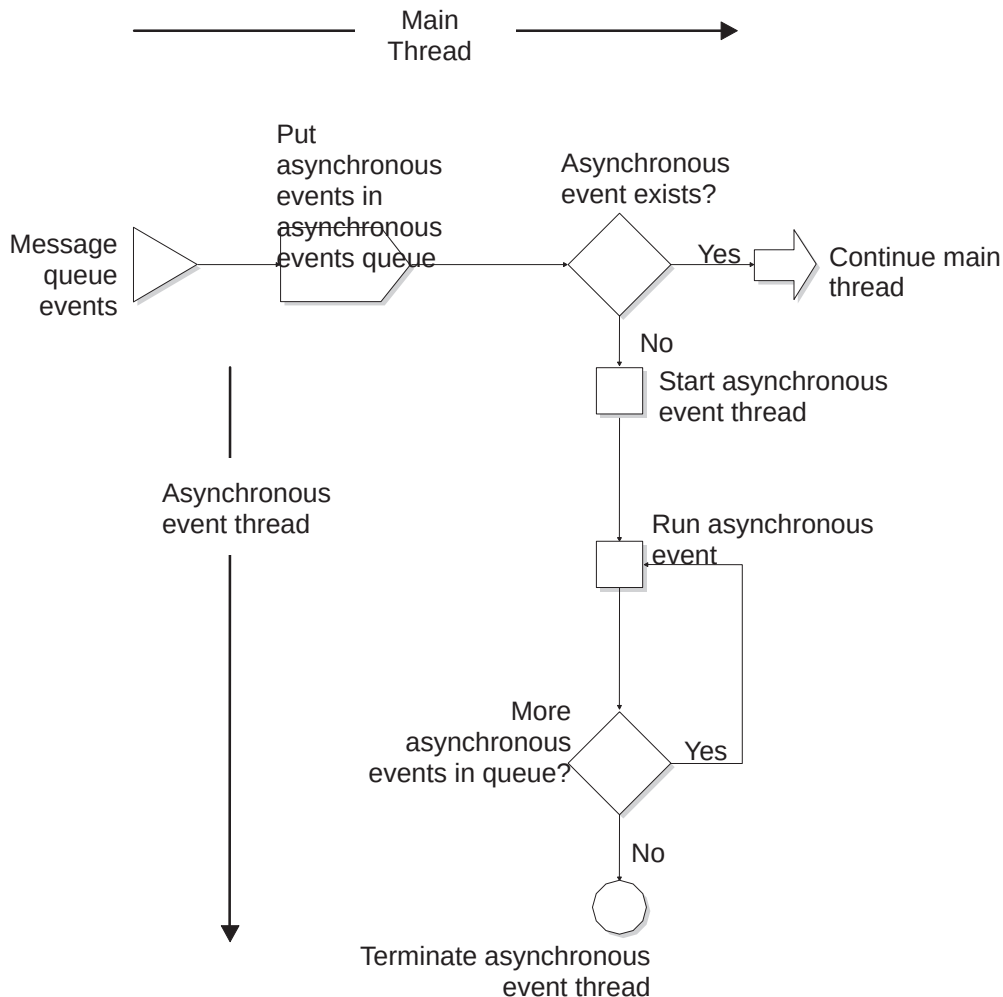
Asynchronous processing does not support form interconnections. Asynchronous processes are used for background processing, and users should not interact with an asynchronous thread directly. For example, if you have an asynchronous process (such as *Row is Exited and Changed - Asynch*) you should not open a form on that worker thread because you are asking the user to interact with a process that should be processing in the background.

This chapter describes the following:

- ☐ Asynchronous events
- ☐ Asynchronous business functions

Asynchronous Events

The following diagram illustrates how threaded events work.



Only one asynchronous event is processed at a time. There might be multiple instances of a unique asynchronous event waiting to process. The main thread continues regardless of the presence or absence of asynchronous events in the queue. Asynchronous event exists in the diagram above determines whether the secondary thread will start; 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 is run on a thread. That means that the user will be allowed to continue keying in data while the event is executing.

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 only available on update grids (Header and Headerless Detail forms).

The first two events will be processed before the user can continue. The *Row is Exited and Changed - Asynch* event is run on a thread. That means that the user can continue entering data while the event is executing. As this event is processing for a specific row, an hourglass is displayed in the row header.

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).

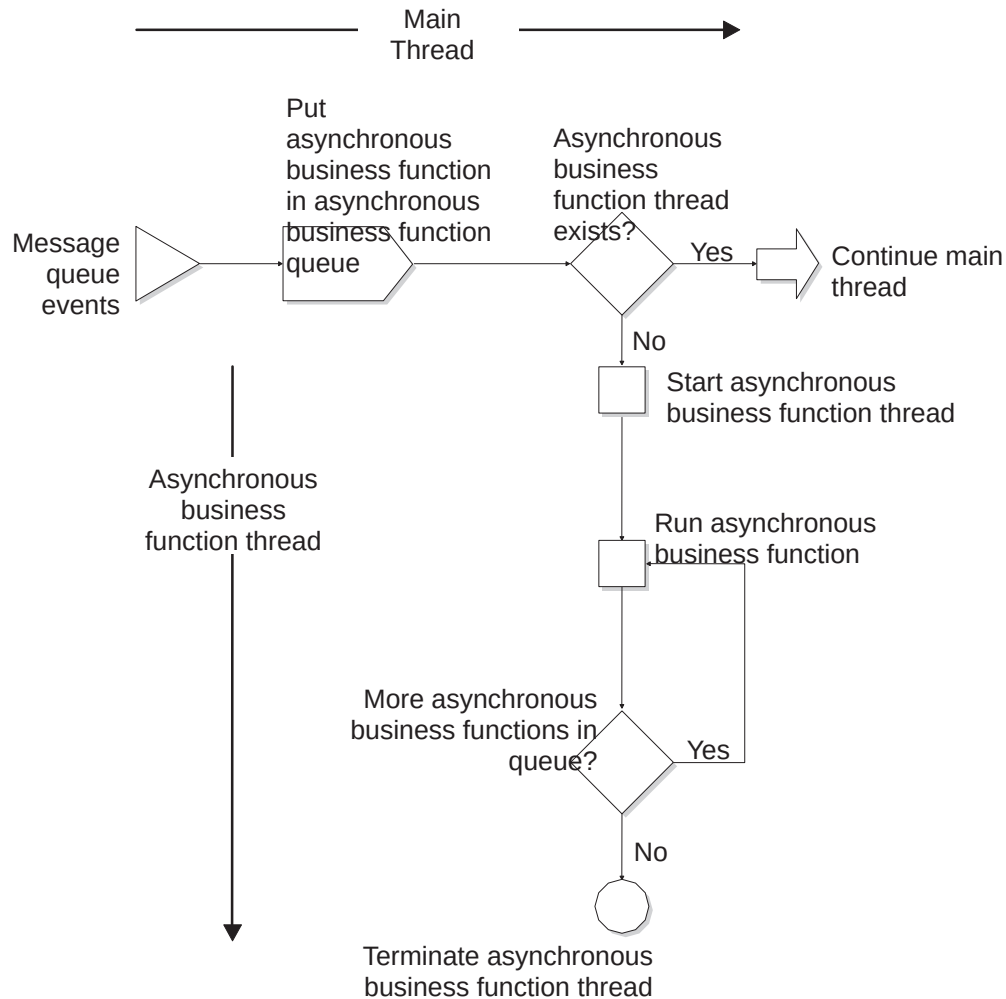
The first two events are processed before the user can continue. The *Column is Exited and Changed - Async* events is run on a thread. That means the user will be allowed to continue keying in data while the event is processing.

Asynchronous Business Functions

Business functions that are run on OK and Cancel processing are a performance bottleneck. Where possible these business functions should be run on a thread. In Event Rules Design there is an asynchronous option on the form where business function parameters are selected. The 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 will process on a thread. When the option is turned on, there are only two possibilities for passing parameters. Parameters can be passed in, or not passed at all. Passing parameters out of the business function is not possible because the form has continued and may not even still be open. Attempts to set errors during asynchronous business functions will be ignored. The asynchronous business function has no connection to the form, so errors cannot be set.

Asynchronous business functions make the most sense for forms that make many database calls as the form is closing. A form that uses jdeCache calls or work files to store records temporarily and then writes the temporary records to the actual files when OK is clicked is a good candidate for using threaded business functions.

The following diagram illustrates how threaded business functions work.



Only one asynchronous business function is processed at a time. The main thread continues regardless of the presence or absence of asynchronous events in the queue. *Asynchronous business function thread exists* in the diagram above determines whether the secondary thread will start, it does not affect the main thread.

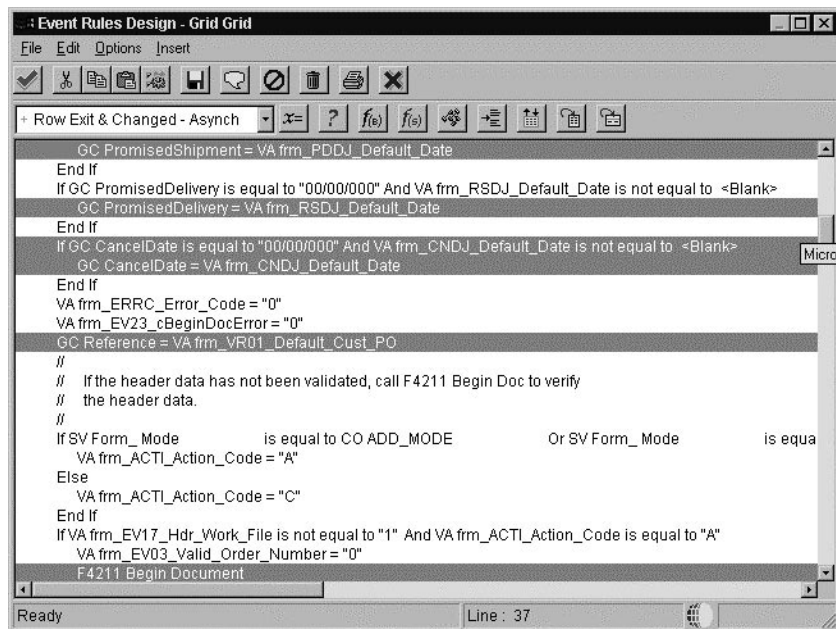
Asynchronous business functions must complete their processing before OneWorld Explorer can be closed. If there are still asynchronous business functions running, a message will be displayed and OneWorld Explorer will not close.

Example: Asynchronous Event Processing

For the *Row is exited and Changed - Asynch* event the following is an example of functions attached.

The *Row is exited and Changed - Asynch* event behind the grid in Sales Order Detail form (P4210) has logic attached to it as follows.

For the first five highlighted lines a lot of default information is being populated in certain table columns for the database update. For the last highlighted line, this is where the F4211Begin Document (Master Business Function) is called.



Working with 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. This is useful for debugging specific event rules.

BrowsER displays the structure of forms within an interactive application, or sections within a batch application. The forms or sections are displayed in a hierarchical structure, with events and event rules for each section.

This chapter describes the following:

- ☐ Working with BrowsER
- ☐ Working with BrowsER options

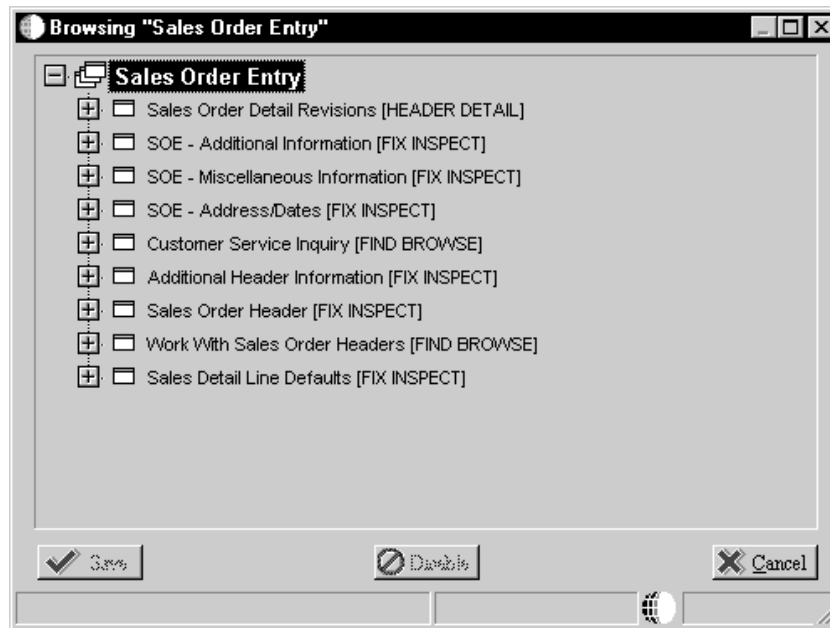
Working with BrowsER

You can use BrowsER to disable or enable event rules, then observe the impact on your application.

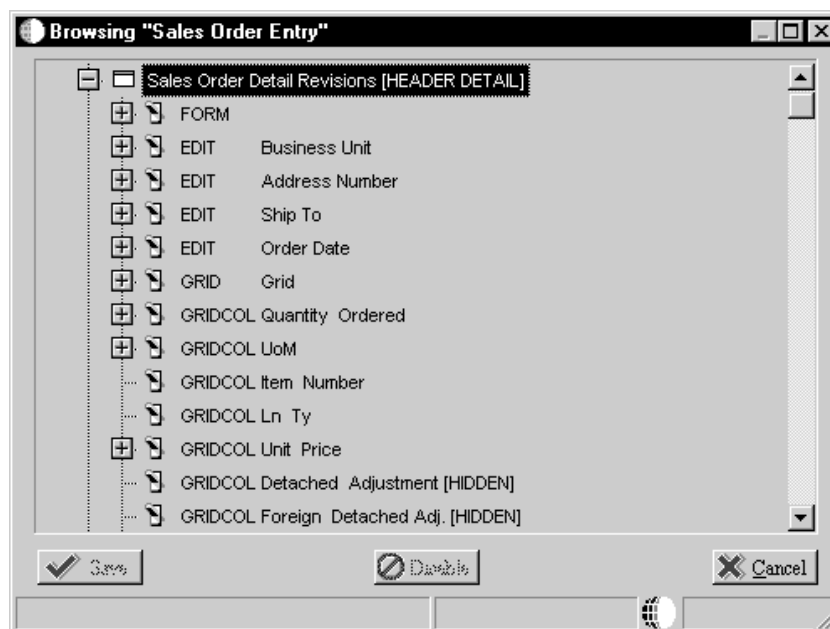


To work with BrowsER

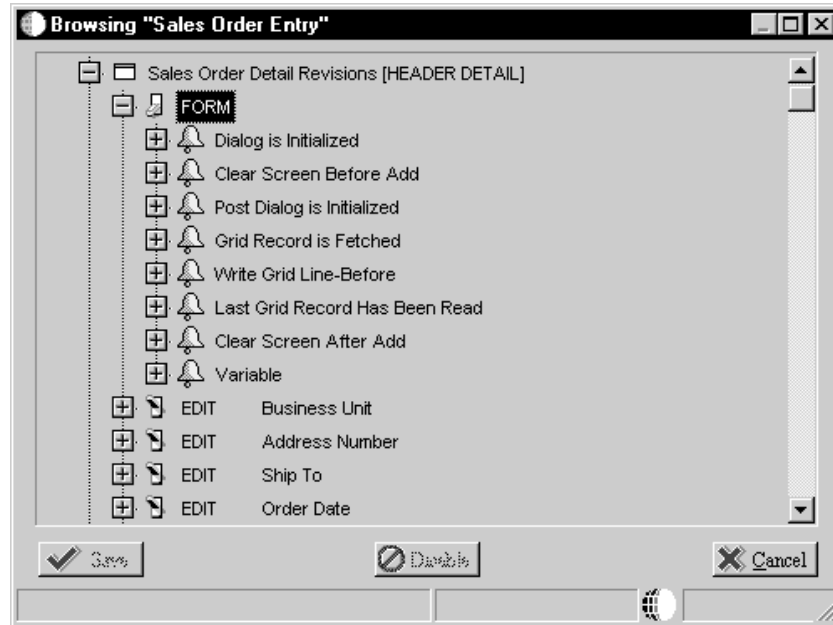
1. Choose one of the following methods to access BrowsER.
 - On Application Design Aid, from the Form menu, choose BrowsER.
 - On Form Design Aid, from the View menu, choose BrowsER.



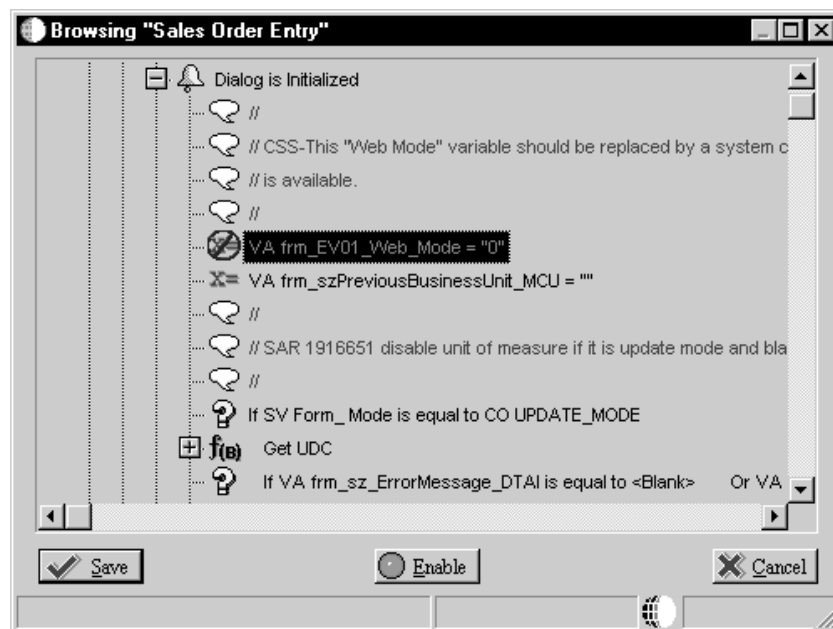
2. On BrowsER, click the plus (+) and minus (-) buttons to expand or collapse the hierarchical view of events for interactive forms or batch report sections.



Each event rule is displayed beneath the event it is associated with and also beside a control that contains event rule logic. If it is not displayed beside a control, then there is no event rule logic on that control.



- To disable an event rule for debugging, double-click on the event rule.



- Double-click a disabled event rule to enable it.

You cannot print or modify any event rule from any BrowsER forms.

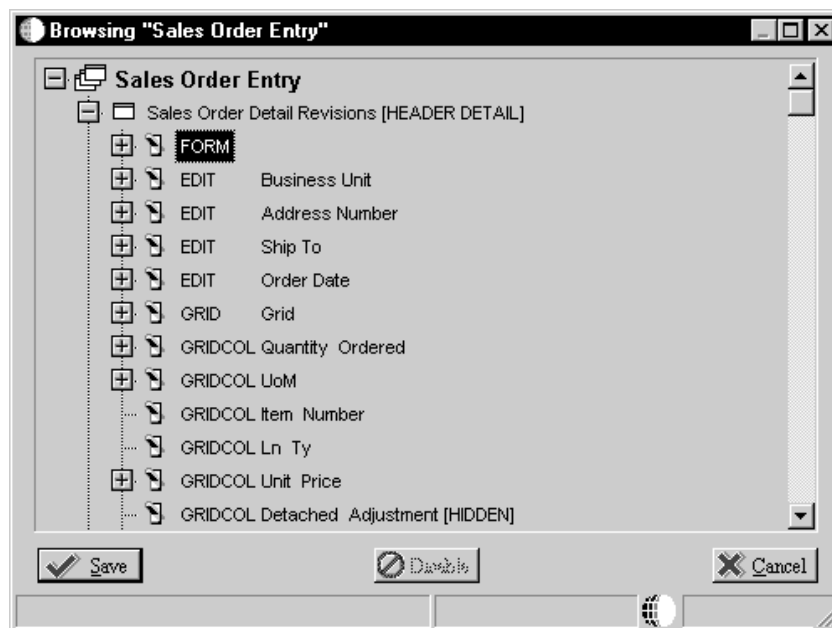
Working with BrowsER Options

You can choose one of several BrowsER options to easily view or search for code, including:

- Expand Tree
- Expand Node
- Show Object IDs
- Hide Objects with no ER
- Filter ER Records
- Search

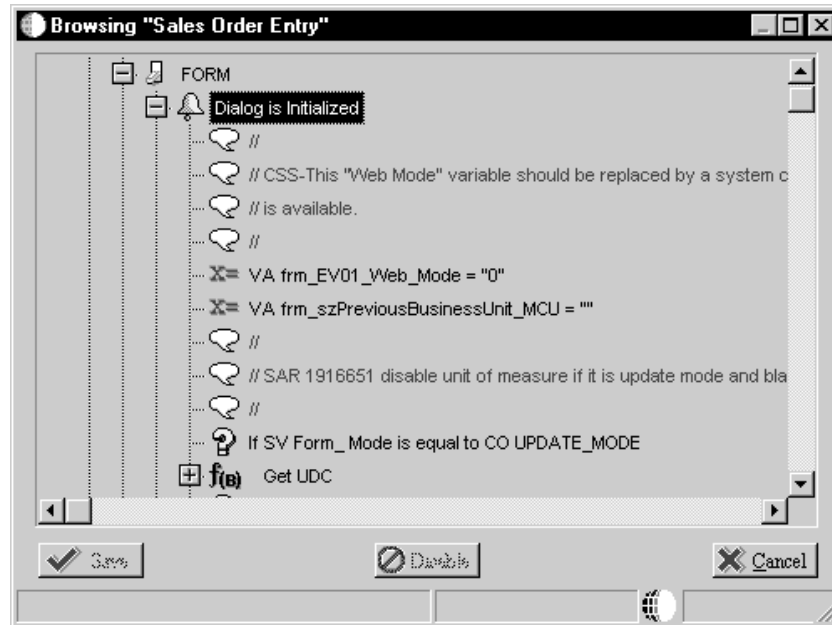
► To work with BrowsER options

On the Browsing form, hold down the right mouse button and choose an option from the menu that appears.



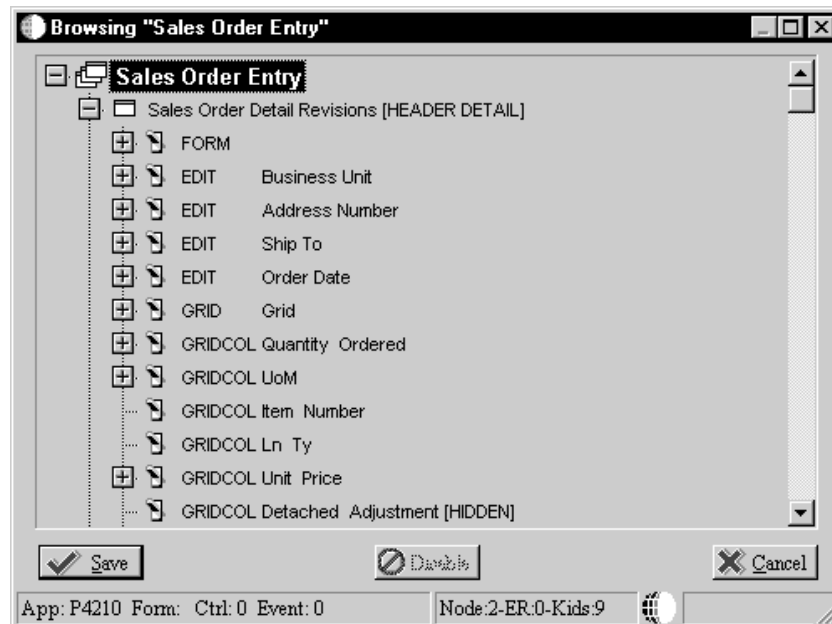
Expand Node

Expand Node allows you to expand a highlighted node to show each line of code for the node.



Show Object IDs

Show Object IDs is used for C coding. It displays a unique ID for each object. This is useful to identify objects that have errors displayed in the code generation process.



Filter ER Records

Filter ER Records allows you to look for certain types of code, including:

- 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 it to find occurrences of particular strings or variables.

Business Functions



Business Functions

You can use business functions to enhance OneWorld applications. Business functions group related business logic. Business functions should perform a specific task. Journal Entry Transactions, Calculating Depreciation, and Sales Order Transactions are examples of cohesive business functions.

You can create business functions using one of the following methods:

- The event rules scripting language

Business functions created using the event rules scripting language are referred to as Business Function Event Rules (also called Named Event Rules). You should try to use Business Function Event rules for your business functions if possible. There may be some instances, however, where C business functions may better suit your needs.

- C programming code

Business functions designed using C are not generated by OneWorld. C Business Functions are used mainly for caching. They can also be used for:

- Batch error level messaging
- Changing the OR properties of a Where clause
- Large functions
- Complex Select statements

C business functions work better for large functions. If you have a large function, you can break the code up into smaller individual functions and call them from the larger function.

After you create business functions, you can attach them to OneWorld applications to provide additional power, flexibility, and control.

This section describes:

- ☐ Creating Business Function Event Rules
- ☐ Understanding C Business Functions
- ☐ Using Application Programming Interfaces
- ☐ Working with Business Functions



- ☐ Creating and Specifying Custom DLLs
- ☐ Working with Business Function Builder
- ☐ Transaction Master Business Functions
- ☐ Master File Master Business Functions
- ☐ Business Function Documentation

Business Function Features

Feature	Description
Data Integrity	Business functions help perform referential integrity of corporate data residing in databases. They also provide additional data integrity beyond what is offered with database vendors, including working with multiple database platforms and distributed databases. OneWorld Tools can perform referential integrity through table I/O and triggers. However, there are performance tradeoffs, so you will sometimes need to write your business functions in C.
Data Manipulation	Business functions can execute standard database manipulation operations. Use the JDEBASE APIs to perform TableOpens, Fetches, and so forth. The APIs provide consistency and an insulation from embedded SQL.
Event Processing	Business functions offer powerful solutions to event programming with graphical user interfaces. OneWorld Tools also provide these solutions.
Runtime Flexibility	Business functions provide runtime flexibility to business solutions. Business functions can be mapped to run on different platforms through OneWorld's Object Configuration Manager (OCM).
Advanced Solutions	Business functions offer advanced processing solutions to a wide range of business issues. They are your "hook" into OneWorld. You can use OneWorld's Imaging APIs to use powerful, full-featured, third-party imaging systems with OneWorld. Business functions allow you a way of protecting custom code during a reinstallation of OneWorld.

Business functions can be distributed on different operating platforms.

What are the Components of a Business Function?

The process of creating a business function produces several components. The Object Librarian is the entry point for the tools that create the components. The following components are created:

Component	Where Created
Business Function Specifications	Object Librarian
	Business Function Source Librarian and Business Function Design
Data Structure Specifications	Object Librarian
	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 have been divided into categories. This distribution provides better separation between the major functional groups: tools, financials, manufacturing, distribution, and so forth. Most business functions are organized into a consolidated DLL based on the business function's system code. For example, a financials business function with system code 01 would be placed in CFIN.DLL.

You should follow these guidelines when you add or modify business functions:

- You should create a custom parent DLL unless you are adding a J.D. Edwards business function. Business functions are assigned a parent DLL based on the system code defined in H92/PL UDC. If no DLL is assigned for the system code that the business function is being created in, CCUSTOM is used. 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. Failure to use `jdeCallObject` will result in linker errors if you are attempting to call a business function in a different DLL. A linker error will prevent your function call from working.

Following are some of the DLLs for which Business Function Builder manages the builds:

DLL Name	Functional Group
CAEC	Architecture
CALLBSFN	Consolidate BSFN Library
CAPS	Advanced Planning and Scheduling
CCONVERT	Conversion Business Functions
CCORE	Core Business Functions
CCRIN	Cross Industry Application
CCUSTOM	Custom Business Functions
CDBASE	Tools – Database
CDDICT	Tools – Data Dicionary
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 – Modificated BFs

DLL Name	Functional Group
CMFGBASE	Manufacturing Base Functions
COBJLIB	Tools – Object Librarian
COBLIB	Busbuild Functions
COPBASE	Distribution/Logistic Base Functions
CRUNTIME	Tools – Run Time
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

Do not use the table triggers for regular business functions.

How Distributed Business Functions Work

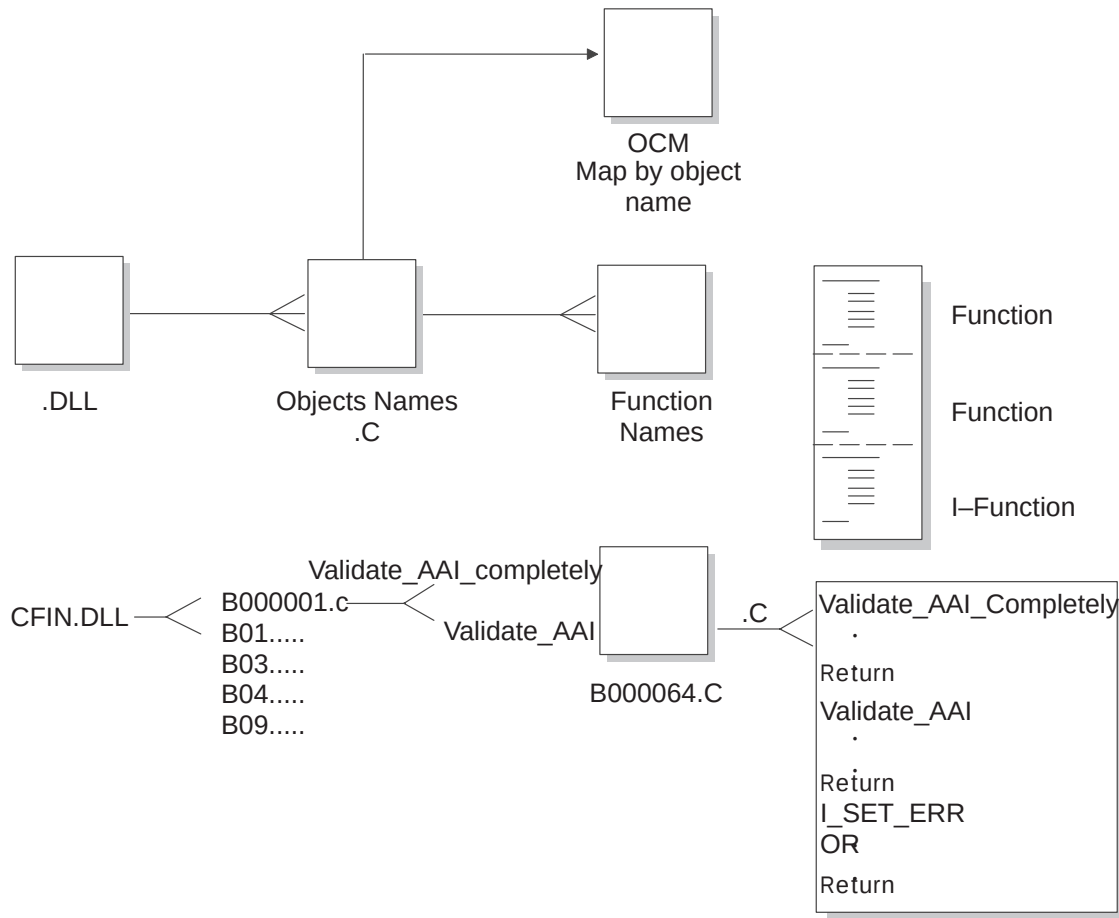
There are three main pieces that make up named event rules or business functions.

- Object Name is controlled through Object Librarian. It is the actual source file.

- Function Name is controlled through Object Librarian. Each function must have a data structure.
- DLL Name is controlled through Object Librarian. The DLL is a dynamic link library.

Object Name (1)

Function Name (1 or more)
DLL Name



When a business function is called, the system refers to the Object Configuration Manager (OCM) to determine where to run the business function. Once a business function is mapped to a server, calls from that business function cannot be mapped back to the workstation. Refer to the *CNC 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 whose source language is event rules, instead of C. It is created using the event rules scripting language. This scripting language is platform independent and is stored in a database as a OneWorld object. Business function event rules are also called named event rules (NER).

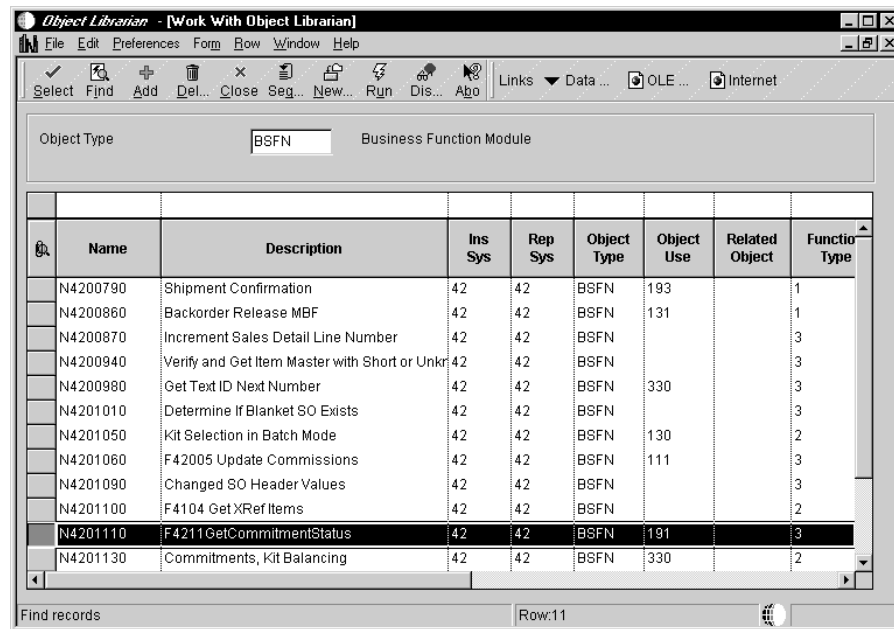
Before you design a business function event rule you must have a data structure. Refer to *Data Structures* in this guide for more information about creating data structures.

To work with a business function event rule you must:

- Design a business function event rule
- Create or edit the event rules for the business function event rules

► To design a business function event rule

On Object Librarian



	Name	Description	Ins Sys	Rep Sys	Object Type	Object Use	Related Object	Function Type
	N4200790	Shipment Confirmation	42	42	BSFN	193		1
	N4200860	Backorder Release MBF	42	42	BSFN	131		1
	N4200870	Increment Sales Detail Line Number	42	42	BSFN			3
	N4200940	Verify and Get Item Master with Short or Unkr	42	42	BSFN			3
	N4200980	Get Text ID Next Number	42	42	BSFN	330		3
	N4201010	Determine If Blanket SO Exists	42	42	BSFN			3
	N4201050	Kit Selection in Batch Mode	42	42	BSFN	130		2
	N4201060	F42005 Update Commissions	42	42	BSFN	111		3
	N4201090	Changed SO Header Values	42	42	BSFN			3
	N4201100	F4104 GetXRef Items	42	42	BSFN			2
	N4201110	F4211GetCommitmentStatus	42	42	BSFN	191		3
	N4201130	Commitments, Kit Balancing	42	42	BSFN	330		2

1. Add a new business function event rule or select an existing one to modify.

Ensure that the source language for the business function object (BSFN) is NER.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PG	1	1492983	AC277173	4/16/97	00:00:00	
INTELNTC	PROD	1	1492983	JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PG	3	1492983	BS5625464	8/4/98	18:23:20	

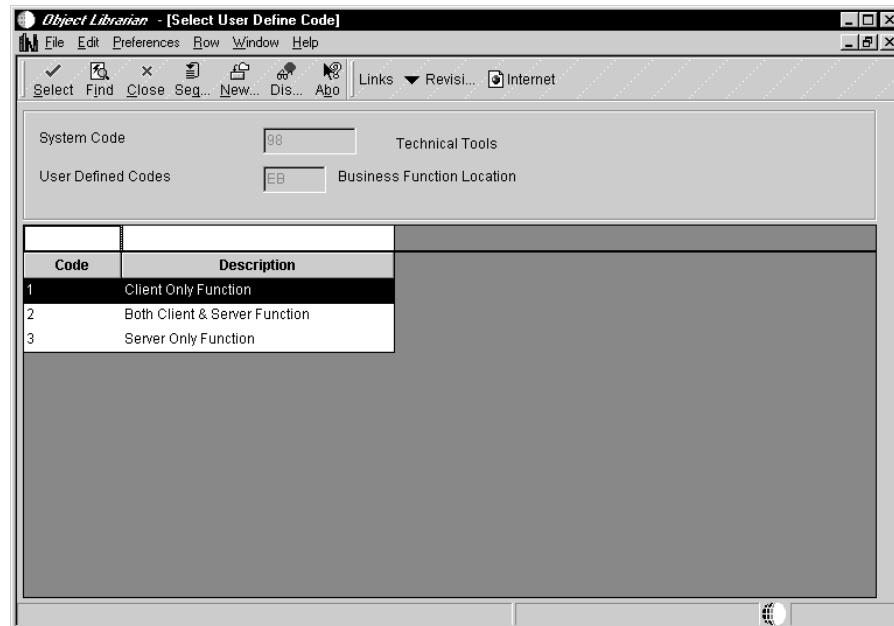
- On Business Function Source Librarian, choose Design from the Form menu.

Function Name	Description	Function Category	F
F4211GetCommitmentStatus	F4211 Get Commitment Status		GET

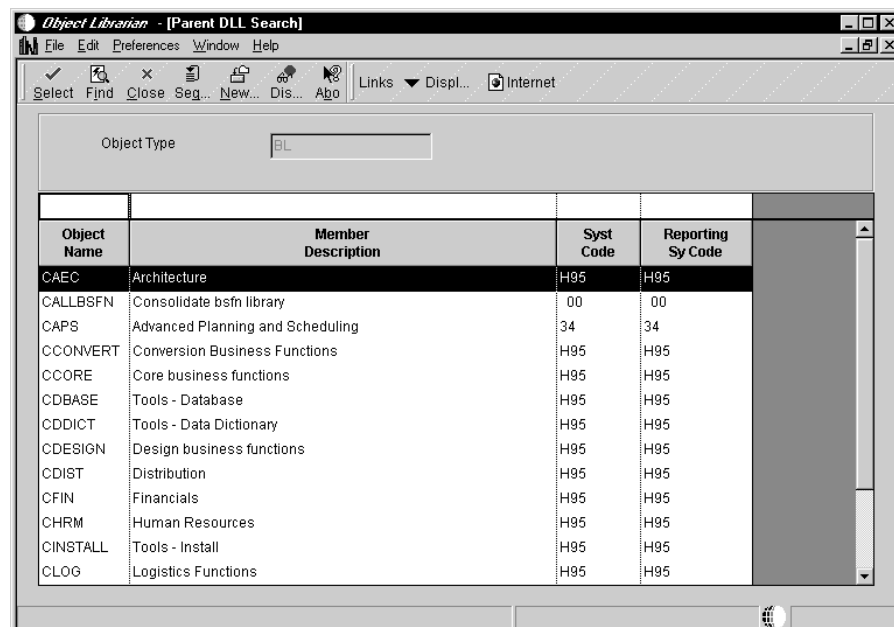
- On Business Function Design, complete the following fields:
 - Source Name
 - Description
- Choose the Parent DLL.

5. Choose the Business Function Location.

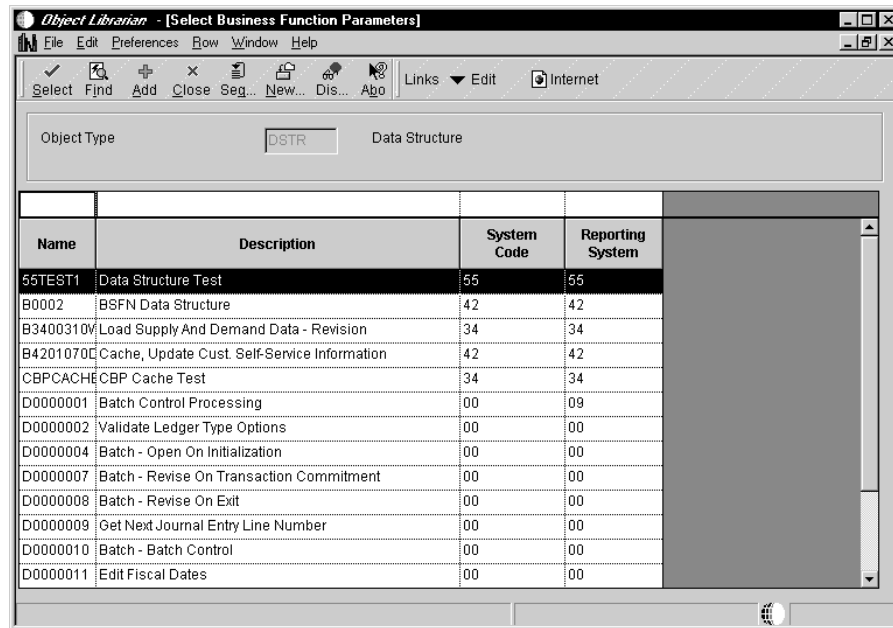
This is where the business function call is built. You typically always choose option 2 for Both Client and Server Function. The Client Only and Server Only function locations are used only if absolutely necessary.



6. Choose the parent library in which you want this business function to reside.



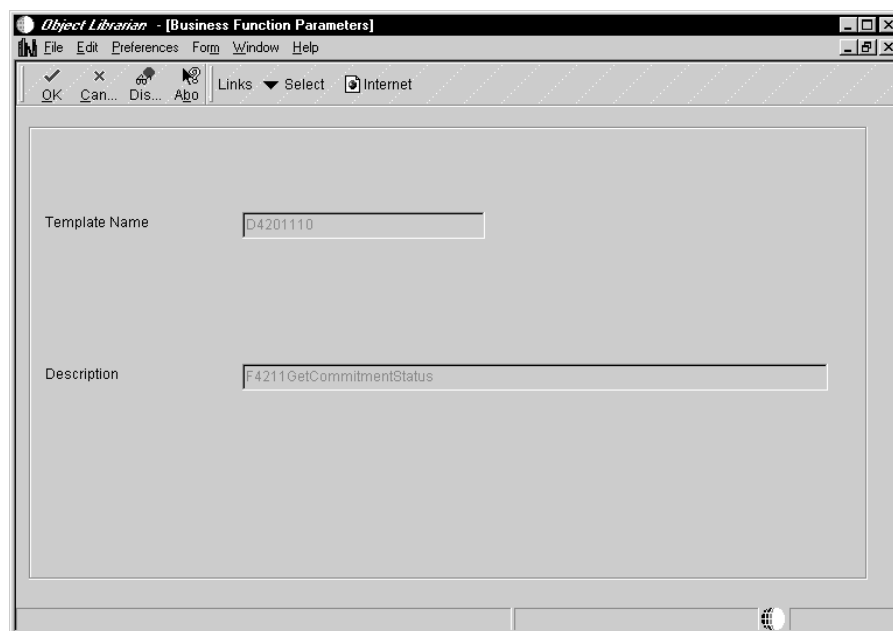
7. Choose Parameters from the Row menu.



8. On Business Function Parameters, choose Select from the Form menu.
9. Choose 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. If you need to do this, click Add. Refer to Data Structures for more information about data structures.

If the business function already has a data structure attached the following form appears. You can choose another data structure or edit the one you want to use.



► To create or edit event rules for a business function event rule

On Business Function Design

1. From the Form menu, choose Edit to create or edit the event rules that will comprise the function.
2. On Business Function Event Rules Design, construct the business function event rule.
3. Click the Save button or OK to save the event rules source code and return to Business Function Design.
4. On Business Function Design, click OK to save the functions and return to Business Function Source Librarian.
5. From the Form menu, choose Build to create the .c and .h files, 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 associated with the business function.

6. 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 browsing available business function objects.

See Also

Attaching Functions in this guide.

Example: Named Event Rule

Named event rules can be reused in multiple places by multiple programs. This modularity lends itself to less rework, streamlining, and reusability of code.

Not all chunks of code should be packaged into a business function module. For example when code is so specific that it only applies to a particular program and is not reused by any other programs, you should leave it in one place. You do not want to package it into a business function. One way to handle this situation is to attach all the logic on a hidden control (*Button Clicked* event) and use a system function to execute the logic as needed.

A good example of a named event rule is N3201030. This business function creates generic text and Work Order detail records (F4802) for the 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 would appear in Event Rules Design.


```
//
// 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:
```

```
// S0 line 13.001 will have 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 C0 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 C0 SUCCESS

VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Cfg 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
Cfg 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
```

Understanding C Business Functions

This section includes information about:

- The layout of standard business function header and source files
- The structure of a business function source file

Header File Sections

The following are the major sections of a business function header file:

Section	What it Includes	Description
Header File Comment	- Header file name - Description - History - Programmer - Software Action Request (SAR) Number - Copyright information	Built by the input process of the Business Function Source Librarian. The programmer name and SAR number are manually updated by the programmer.
Table Header Inclusions	Include statements for header files associated with tables 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 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	Enter symbolic names in uppercase, separating words by an underscore. Use sparingly.
Structure Type Definitions	Data structure definitions for internal processing	Define this structure, making sure that structure names are prefixed by the source file name to prevent naming conflicts.
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 Librarian.
Source Preprocessor	- Undefined 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 what business functions are in the source file, what parameters are being passed to them, and what type of value they return
Internal Function Prototype	Prototypes for all internal functions required to support business functions within this source file	Defines what internal functions are associated with the business functions in the source file, what parameters are being passed to each internal function, and what type of value they return

Business Function Header File Example

The following header file was created by Business Function Design. It contains the minimum components required in a business function header file. Numbers correspond to the explanation provided at the end of the example.

```

1.  /*****
    *   Header File:  B98SA001.h
2.  *
    *   Description:  Check for In Add Mode Header File
    *
    *   History:
3.  *       Date          Programmer  SAR# - Description
    *       -----          -
    *   Author 03/07/1995   Hotchkiss  Unknown - Created
    *
    *   Copyright (c) 1994  J.D. Edwards & Company
    *
    *   This unpublished material is proprietary to J.D. Edwards & Company.
    *   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 J.D. Edwards & Company.
    *****/

4.  #ifndef __B98SA001_H
    #define __B98SA001_H
    '
5.  * Table Header Inclusions
    *****/

6.  /*****
    * External Business Function Header Inclusions
    *****/

7.  /*****
    * Global Definitions
    *****/

8.  /*****
    * Structure Definitions
    *****/

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

9.  /*****
    * TYPEDEF for Data Structure
    *   Template Name: Check for In Add Mode
    *   Template ID:   104438
    *   Generated:     Tue Mar 07 09:33:47 1995
    *
    * DO NOT EDIT THE FOLLOWING TYPEDEF
    *   To make modifications, use the OneWorld Data Structure
    *   Tool to Generate a revised version, and paste from
    *   the clipboard.
    *****/

    #ifndef DATASTRUCTURE_104438
    #define DATASTRUCTURE_104438

```

```

typedef struct tagDS104438
{
    char                cEverestEventPoint01;           /* OneWorld Event Point 01 */
} DS104438, FAR *LPDS104438;

#define IDERRcEverestEventPoint01_1                    1L

#endif

/*****
10. * Source Preprocessor Definitions
    *****/
#define JDEBFRTN
#define JDEBFRTN(r) __declspec(dllexport) r
#define JDEBFRTN(r) __declspec(dllimport) r
#define JDEBFRTN(r) r
#define JDEBFRTN(r) r

/*****
11. * Business Function Prototypes
    *****/
12. JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode
    (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDS104438 lpDS);

/*****
13. * Internal Function Prototypes
    *****/

#endif /* __B98SA001_H */

```

	Where Input	Description
1.	Object Librarian	Verify the name of the business function header file.
2.	Object Librarian	Verify the description.
3.	IDE	Manually update the modification log with the programmer name and the appropriate SAR number.
4.	Business Function Design	Symbolic constant keeps the contents from being included multiple times.
5.	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.	Business Function Design	No external business functions for this application.
7.	IDE	Constants and definitions for the business function. J.D. Edwards does not recommend using this block. Global variables are not recommended. Global definitions go in .c not .h.
8.	IDE	Data structures for passing information between business functions, internal functions, and database APIs.

	Where Input	Description										
9.	Business Function Design	Data structure type definition. Used to pass information back and forth between an application or report and a business function. It is placed on the Clipboard and pasted into the header file by the programmer. Its components are: <table><tr><td>Comment Block</td><td>Describes the data structure</td></tr><tr><td>Preprocessor Directives than once</td><td>Ensures the data type is not defined more than once</td></tr><tr><td>Typedef</td><td>Defines the new data type</td></tr><tr><td>#define</td><td>Contains the ID to be used in processing if the related data structure element is in error</td></tr><tr><td>#endif</td><td>Ends the definition of the data structure type definition and its related information</td></tr></table>	Comment Block	Describes the data structure	Preprocessor Directives than once	Ensures the data type is not defined more than once	Typedef	Defines the new data type	#define	Contains the ID to be used in processing if the related data structure element is in error	#endif	Ends the definition of the data structure type definition and its related information
Comment Block	Describes the data structure											
Preprocessor Directives than once	Ensures the data type is not defined more than once											
Typedef	Defines the new data type											
#define	Contains the ID to be used in processing if the related data structure element is in error											
#endif	Ends the definition of the data structure type definition and its related information											
10.	Business Function Design	All business function header files contain this section to ensure the business function is prototyped and declared based on where this header is being included.										
11.	Business Function Design	Used for prototypes of the business function										
12.	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 will vary between business functions. Parameters: LPBHVRCOM: Pointer to a data structure used for communication with business functions. Values include an environment handle. LPVOID: Pointer to a void data structure. Currently used for error processing and 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.										
13.	Business Function Design	Internal function prototypes required to support the business functions in this source file										

What is the Structure of a Business Function Source File?

The Object Librarian builds a template for the business function source file. The business function source file consists of several major sections.

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 given in the Business Function Source Librarian. The programmer name and SAR number are manually updated by the programmer.
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 forth
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	Should be copied and values placed in the specified sections to describe the internal function. Internal function source code must follow the comment block.
Internal Function Source Code	Source code for the internal function described in the comment block	Input by the business function developer as needed. Must be preceded by a populated internal function comment block.

Example: Business Function Source File Check for In Add Mode

The following source file was created by Business Function Design. It contains the minimum components required in a business function source file. The source code in the Main Processing section is manually entered and varies from business function to business function. All other components are completely generated by Business Function Design. Numbers correspond to the explanation provided at the end of the example.


```

1. #include <jde.h>
2. #define b98sa001_c
3. /*****
   *   Source File:  B98SA001.c
   *
   *   Description:  Check for In Add Mode Source File
   *
   *   History:
   *       Date          Programmer  SAR# - Description
   *       -----
   *       Author 03/07/1995 Hotchkiss Unknown - Created
   *
   *   Copyright (c) 1994 J.D. Edwards & Company
   *
   *   This unpublished material is proprietary to J.D. Edwards & Company.
   *   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 J.D. Edwards & Company.
   *****/
4. /*****
   * Notes:
   *****/
5. #include <B98SA001.h>
6. /*****
   *   Business Function:  CheckForInAddMode
   *
   *   Description:  Check for In Add Mode
   *
   *   Parameters:
   *       LPBHVRCOM      lpBhvrCom      Business Function Communications
   *       LPVOID         lpVoid         Void Parameter - DO NOT USE!
   *       LPDS104438     lpDS          Parameter Data Structure Pointer
   *****/
7. JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid,
8. LPDS104438 lpDS)
9. {
10.     /*****
11.      * Variable declarations
12.      *****/
13.     /*****
14.      * Declare structures
15.      *****/
16.     /*****
17.      * Declare pointers
18.      *****/
19.     /*****
20.      * Check for NULL pointers
21.      *****/
22.     if ((lpBhvrCom == NULL) ||
23.         (lpVoid == NULL) ||
24.         (lpDS == NULL))
25.     {
26.         jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", 0);
27.         return ER_ERROR;
28.     }
29.     /*****
30.      * Set pointers
31.      *****/
32.     /*****
33.      * Main Processing
34.      *****/

```

```

14.  /*****
      * Function Clean Up
      *****/
      return (ER_SUCCESS);
    }

15.  /* Internal function comment block */
    /*****
      * Function: Ixxxxxxx_a // Replace "xxxxxxx" with a source file number
                           // and "a" with the function name
      *
      * Notes:
      *
      * Returns:
      *
      * Parameters:
      *****/

```

	Where input	Description
1.	Business Function Design	Includes all base OneWorld definitions.
2.	Business Function Design	Ensures related header file definitions are properly made for this source file.
3.	Object Librarian	Verify the information in the file comment section. Enter the programmer's name, SAR number, and description.
4.	Object Librarian	The header file for this application.
5.	Business Function Design	Verify the name and description in the business function comment block.
6.	Business Function Design	The header of a business function declaration.
7.	IDE	Declare variables local to the business function.
8.	IDE	Declare local data structures to communicate between business functions, internal functions, and the database.
9.	IDE	Declare pointers.
10.	Business Function Design	Business Function Standard Verifies that all communication structures between an application and the business function are valid.
11.	Business Function Design	jdeErrorSet (lpBhvrCom, lpVoid, ID(0), "4363",0) Sets the standard error to be returned to the calling application when any of the communication data structures are invalid.
12.	IDE	Declare and assign pointers appropriate values.
13.	IDE	Provide main functionality for a business function.
14.	IDE	Free any dynamically allocated memory here.
15.	IDE	Define internal functions 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 in the format shown in the example.

Using Application Programming Interfaces (APIs)

Application programming interfaces (APIs) are routines that perform predefined tasks. J.D. Edwards APIs make it easier for third-party applications to interact with OneWorld. These APIs are functions provided to manipulate OneWorld data types, provide common functionality, and access the database. There are several categories of APIs, including the Common Library Routines and J.D. Edwards Database (JDEBASE) APIs.

Programs using the OneWorld APIs are flexible for the following reasons:

- No code modifications are required as functionality is upgraded
- When a J.D. Edwards 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, typically, business functions simply have to be recompiled and relinked.

Using Common Library APIs

The common library APIs consist of APIs that are specific to J.D. Edwards functionality, such as determining whether foreign currency is enabled, manipulating the date format, retrieving link list information, or retrieving math numeric and date information. You can use these APIs for setting up data for calling APIs and modifying data after API calls. Some of the more commonly used categories of APIs include MATH_NUMERIC, JDEDATE, and LINKLIST. There are also miscellaneous other Common Library APIs.

OneWorld-Specific Data Types

There are two main data types provided by OneWorld that you should be concerned with when you create business functions. They are MATH_NUMERIC and JDEDATE. It is always possible that these data types may change. For that reason, it is critical that the Common Library APIs provided by OneWorld are used to manipulate variables of these data types.

MATH_NUMERIC Data Type

The MATH_NUMERIC data type is used exclusively to represent all numeric values in OneWorld. 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
{
    char    String [MAXLEN_MATH_NUMERIC + 1];
    char    Sign;
    char    EditCode;
    short   nDecimalPosition;
    short   nLength;
    WORD    wFlags;
    char    szCurrency [4];
    short   nCurrencyDecimals;
    short   nPrecision;
};

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

MATH_NUMERIC Element	Description
String	The digits without separators
Sign	A minus sign indicates the number is negative, otherwise the value is 0x00.
EditCode	The data dictionary edit code used to format the number for display.
nDecimalPosition	The number of digits from the right to place the decimal.
nLength	The number of digits in the String.
wFlags	Processing flags.
szCurrency	The currency code.
nCurrencyDecimals	The number of currency decimals.
nPrecision	The data dictionary size.

JDEDATE Data Type

The JDEDATE data type is used exclusively to represent all dates in OneWorld. 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;
```

JDEDATE Element	Description
nYear	The year. (four digits)
nMonth	The month.
nDay	The day.

Using Database APIs

OneWorld supports multiple databases. A OneWorld application can access data from a number of databases. APIs provide the following:

- A standard interface to multiple database management systems
- Minimal knowledge of SQL to perform complex database operations
- Add, modify, delete, and query operations on database systems
- Management of memory areas for passing data to and from a database
- Runtime creation and execution of SQL statements
- Improved flexibility over embedded SQL
- Improved method of developing applications in a client/server environment
- Standard return codes from function calls

Understanding Standards and Portability

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

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

Ideally, industry standards should allow users to work identically with different relational database systems. The issue is that each major vendor supports industry standards but also offers extensions to enhance the functionality of the SQL language. Vendors are also constantly releasing 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 without

being impacted by differences between database vendors. When vendors provide a new release, the impact on existing applications needs to be minimal. To solve many of these portability issues, many organizations use standard database interfaces called open database connectivity (ODBC).

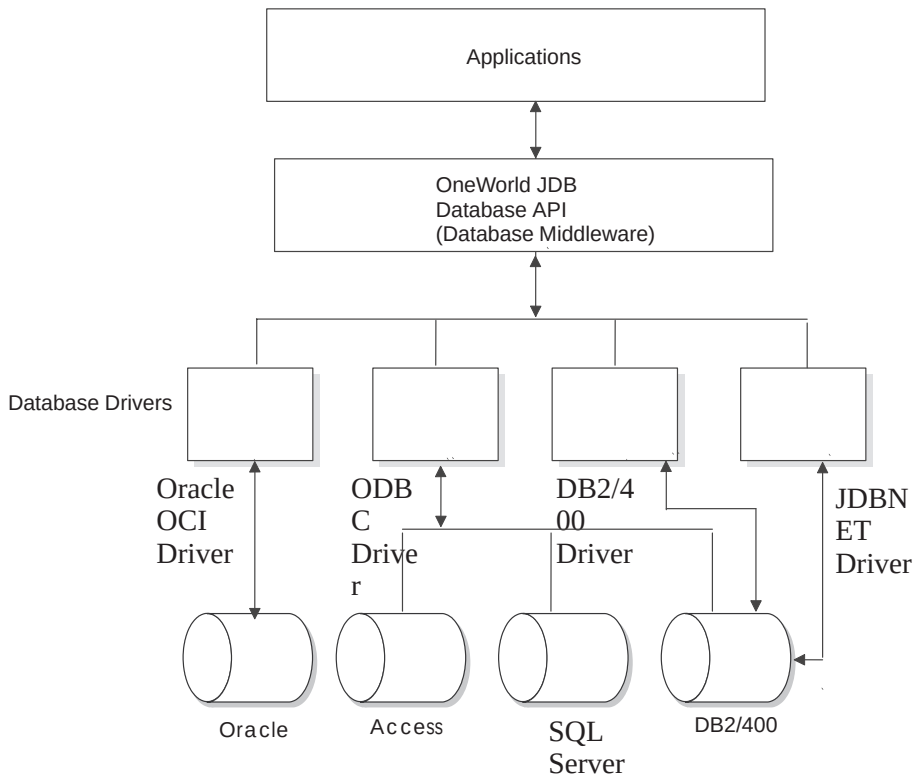
J.D. Edwards Open Database Connectivity (ODBC)

J.D. Edwards ODBC is designed to provide a single access method to multiple relational database management systems. This allows you to use one set of functions to interface with different types of databases. You can develop and compile applications without having to know what type of database the application will be using. Database drivers are installed that allow the J.D. Edwards 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 is also responsible for handling the input/output (I/O) buffers to the database. This allows a programmer to write an application to communicate with a generic data source. It is the responsibility of the database driver to process the API request and communicate with the proper 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 J.D. Edwards database function call. The driver manager will process the request or pass it to an appropriate driver.

The following diagram illustrates how OneWorld uses database APIs.



- OneWorld applications access data from heterogeneous databases
- JDB uses the JDB API to interface between OneWorld applications and multiple databases.
- OneWorld applications and business functions use the JDB API to dynamically generate platform specific SQL statements.
- JDB also supports additional functionality such as replication and cross data source joins.

Standard JDEBASE API Categories

Categories	What it Does
Control Level	Provides initialization and termination level functions for the database connection
Request Level	Provides functions for performing database transactions. The request level functions: 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	Perform and modify information for columns and tables
Global Table/Column Specifications	Provides functionality for creating and manipulating column specifications

Control and request level APIs are used to develop and test business functions.

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 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:

Handle	What it Does
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 a SQL statement. Each request handle is associated with a user handle.

Understanding Database Communication Steps

Several APIs are used 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

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 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

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 may change slightly depending on the calling convention. This information should be included in the documentation for the dll. If you do not know what the calling convention of the dll is, executing 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 some numbers, the dll uses the stdcall calling convention, otherwise it uses cdecl.

The following code example is standard code for windows programs. It is not OneWorld specific.

Stdcall Calling Convention

```
#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 appende 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
```

Cded Calling Convention

The process for using the cded calling convention is similar to the process for using the std calling convention. The main difference is the second parameter for "GetProcAddress" Read the comments preceding that call.

ifdef JDENV_PC

HINSTANCE hLibrary = LoadLibrary(_TEXT("YOUR_LIBRARY.DLL")); // substitute the name of the external dll

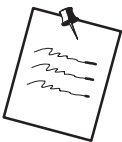
```
if(hLibrary)
{
    // create a typedef for the function pointer based on the parameters and return type of the function to
    // be called. This information can be obtained
    // from the header file of the external dll. The name of the function to be called in the following code
    // is "StartInstallEngine". 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
```



These calls only work on a windows client machine. LoadLibrary & GetProcAddress are windows APIs. If the business function is compiled on a non-windows server, it

will generate errors. Because you cannot run this business function on a non-windows server, you must wrap the code within `#ifdef JDENV_PC` to make your business function compilable on the server.

Calling a Visual Basic Program from OneWorld

You can call a Visual Basic program from OneWorld business function and passing a parameter from the Visual Basic program to the OneWorld business function.

- You must write the Visual Basic program into a Visual Basic DLL that exports the function name of the program, and returns a parameter back to the OneWorld business function.
- Next you must write a business function that loads the Visual Basic DLL using the win32 function `LoadLibrary`.
- In the business function you create, call the win32 function `GetProcAddress` to get the Visual Basic function and call it.

The following code example illustrates how to use the `LoadLibrary` and `GetProcAddress`.

```

/*****
* Business Function: CallVendorSpecificFunctionIn
*
* Description: Call Vendor-Specific Function – Inbound
*
* Parameters:
*   LPBHVRCOM      lpBhvrCom  Business Function Communications
*   LPVOID         lpVoid     Void Parameter – DO NOT USE!
*   LPDSD0000192B lpDS       Parameter Data Structure Pointer
*
*****/

JDEBFRTN (ID) JDEBFWINAPI CallVendorSpecificFunctionIn (LPBHVRCOM lpBhvrCom,
                                                         LPVOID lpVoid,
                                                         LPDSD0000192B lpDS)
{
/*****
* Variable declarations
*****/

ID          iReturnValue      = ER_SUCCESS;

char        szUserId[11]      = { 0 };
char        szBatchNumber[16] = { 0 };
char        szTransactionNumber[23] = { 0 };
double      dLineNumber       = 0.0;
char        cSuccessfullyProcessed = '\0';

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

```

```

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

HINSTANCE      hDLLInstance = (HINSTANCE) NULL;

VendorFnctIn_Proc VendorFnctIn = (VendorFnctIn_Proc) NULL;

/*****
* Check for NULL pointers
*****/
if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
    (lpVoid == (LPVOID) NULL) ||
    (lpDS == (LPDSD0000192B) NULL))
{
    jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
    return ER_ERROR;
}

/*****
* Set pointers
*****/

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

/* Set return flag */
lpDS->cErrorsEncountered = '\0';

/* Load vendor function parameters */
strcpy (szUserId, (const char *) lpDS->szUserId);
strcpy (szBatchNumber, (const char *) lpDS->szBatchNumber);
strcpy (szTransactionNumber, (const char *) lpDS->szTransactionNumber);
dLineNumber = MathNumericToDouble(&lpDS->mnLineNumber);
cSuccessfullyProcessed = '\0';

/* Load the library */
hDLLInstance = LoadLibrary(lpDS->szFunctionLibrary);

/* Error handling for problem in loading library */
if (hDLLInstance == (HINSTANCE) NULL)
{
    jdeErrorSet (lpBhvrCom,
                lpVoid,
                (ID) 0,
                "096G",
                (LPVOID) NULL);
    lpDS->cErrorsEncountered = '1';

    iReturnValue = ER_ERROR;
}
else
{
    /* Retrieve the address of the vendor function */
    VendorFnctIn = (VendorFnctIn_Proc)

```

```

        GetProcAddress(hDLLInstance, lpDS->szFunctionName);

/* Error Handling for not loading the function */
if (VendorFunctIn == (VendorFunctIn_Proc) NULL)
{
    jdeErrorSet (lpBhvrCom,
                lpVoid,
                (ID) 0,
                "096G",
                (LPVOID) NULL);
    lpDS->cErrorsEncountered = '1';

    iReturnValue = ER_ERROR;

}
else
{
    /* Call the vendor function */
    VendorFunctIn (szUserId,
                  szBatchNumber,
                  szTransactionNumber,
                  dLineNumber,
                  cSuccessfullyProcessed);
}
} /* End if for retrieving function addresses */

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

if (hDLLInstance)
{
    FreeLibrary (hDLLInstance);
}

return (iReturnValue);

```


Working with Business Functions

Every business function must follow a defined structure and form. Every line of code must conform to the J.D. Edwards business function programming standards. Following these standards will allow you to take advantage of a proven software engineering approach for developing applications:

- Use Object Librarian to build business function data structures
- Use Object Librarian to create business function source and header files
- Create and add data structure type definitions to the header file

This chapter describes:

- Viewing Object Codes
- Adding Related Tables and Functions
- Debugging Business Functions
- Changing a Business Function Parent DLL

Viewing Object Codes

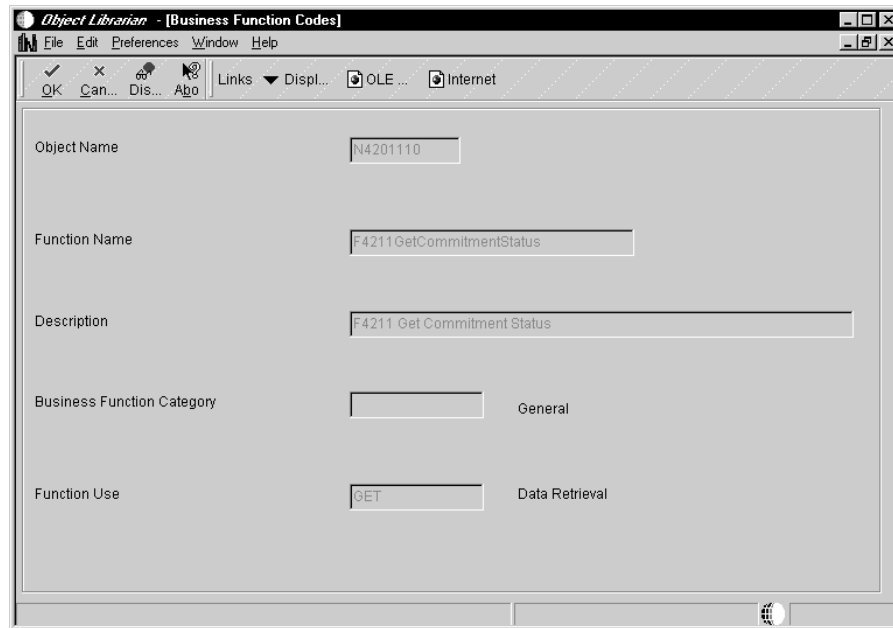
Object codes are used to group business functions into categories. These codes can then be used as filters for searches.



To view object codes

From Business Function Design

1. From the Row menu, choose Codes.



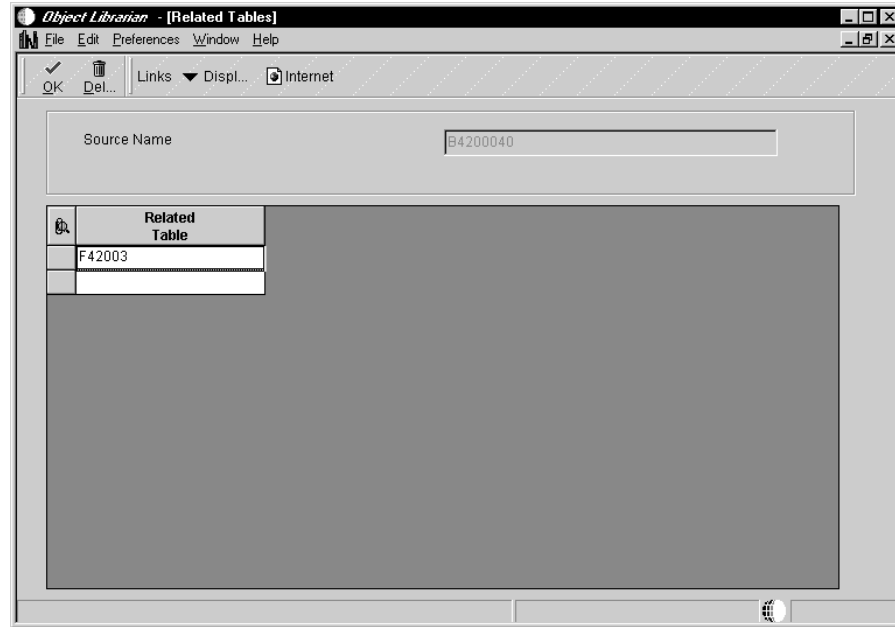
Adding Related Tables and Functions

Both tables and functions can be attached 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 and functions

On Business Function Design

1. From the Form menu, choose Tables.



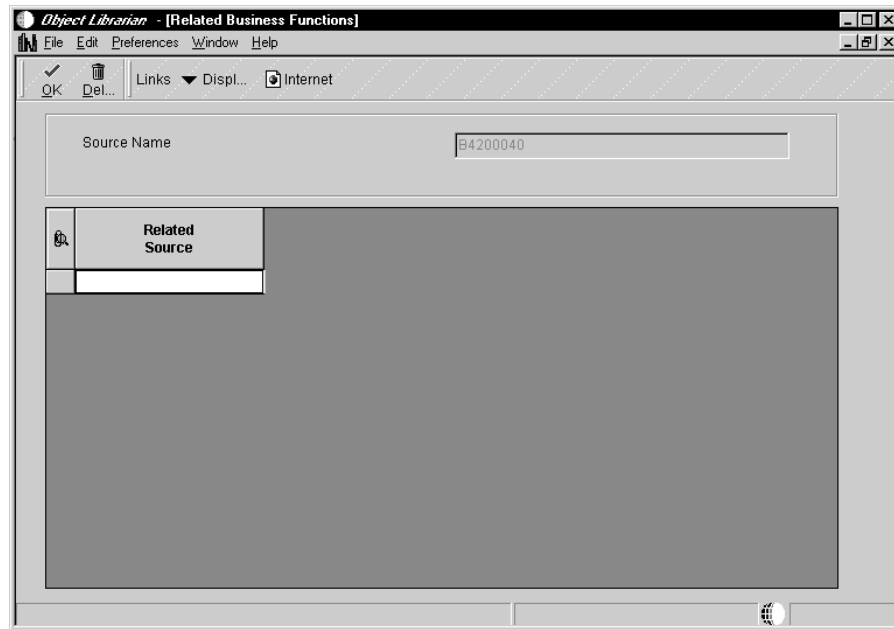
2. Enter the names of related tables.
3. Click OK.

The associated record is stored in the F9863 table.

► To add related functions

From Business Function Design

1. From the Form menu, choose Functions.



2. Enter the names of related business function source object names.
3. Click OK.

The associated record is stored in the F9863 table.

Debugging Business Functions

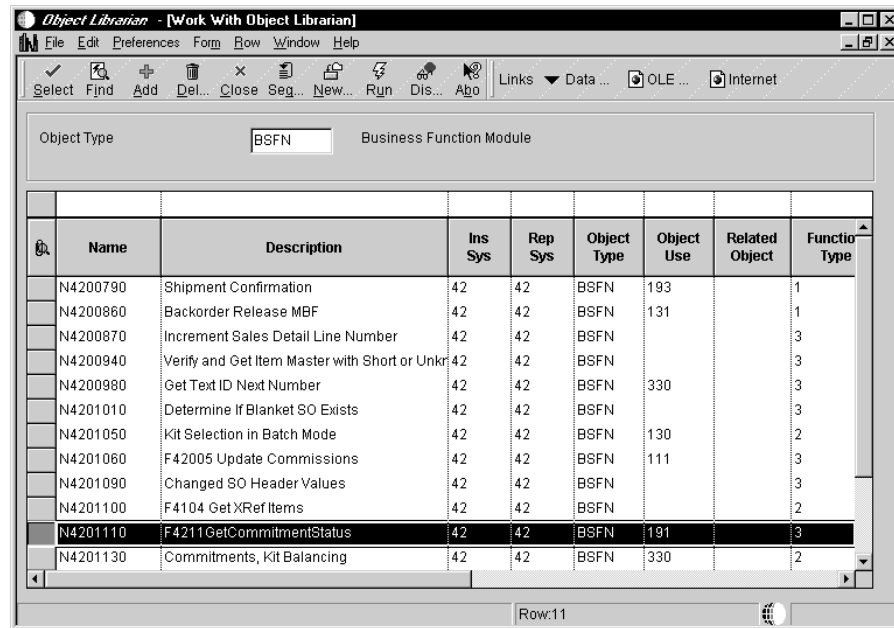
For information about debugging business functions, refer to *Debugging* in this guide. Because the source code for business function event rules is generated into C, follow the same procedures for debugging both C and NER business functions.

Changing a Business Function Parent DLL or Location

You may need to change the parent DLL for a business function.

► To change a business function parent DLL

On Object Librarian



1. Check out the business function.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1492983	AC277173	4/16/97	00:00:00	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PGF	3	1492983	BS5625464	8/4/98	18:29:00	

- From the Form menu, choose Design.

Function Name	Description	Function Category	F
F4211GetCommitmentStatus	F4211 Get Commitment Status		GET

- On Business Function Design, type the new DLL name in Parent DLL, or change the business function location and click OK.
- Check in the business function.

You should transfer this business function to all path codes as soon as possible.

Your change is available to everyone in the next subsequent package. Refer to the *Package Builds Guide* for information about custom workstation DLLs.

Creating and Specifying Custom DLLs

Business function DLLs are consolidated. Therefore, for your custom business functions, you need to build each one into a custom DLL that you created. This ensures that your custom business functions stay separate from J.D. Edwards business functions. The build program looks at the Object Librarian header file (F9860) to verify that your custom DLL exists.

To create and specify a custom DLL, see the following:

- ☐ Creating a custom DLL
- ☐ Specifying a custom DLL for a custom business function

Creating a Custom DLL

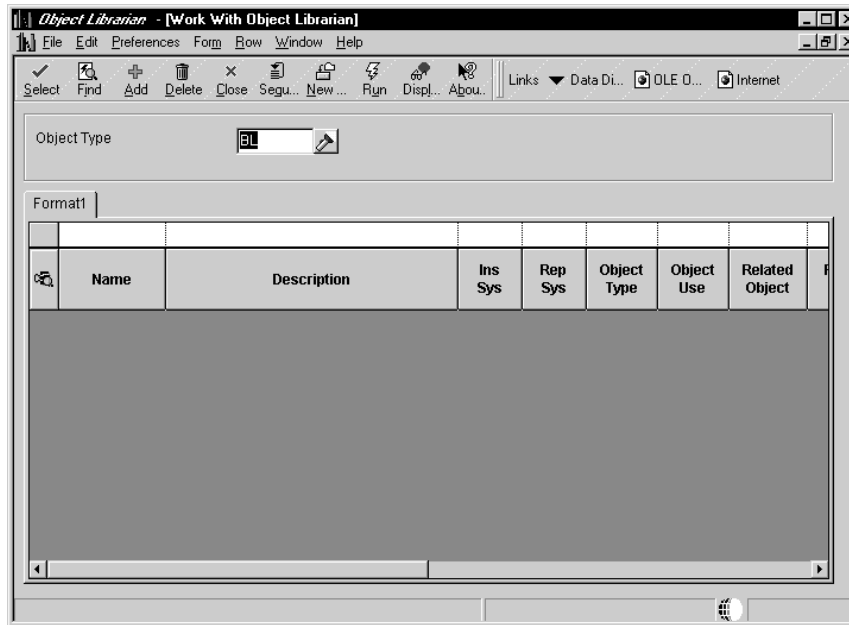
Before you can build a custom business function into a custom DLL, that custom DLL must exist in the Object Librarian header file (F9860). Use this task to create a custom DLL.



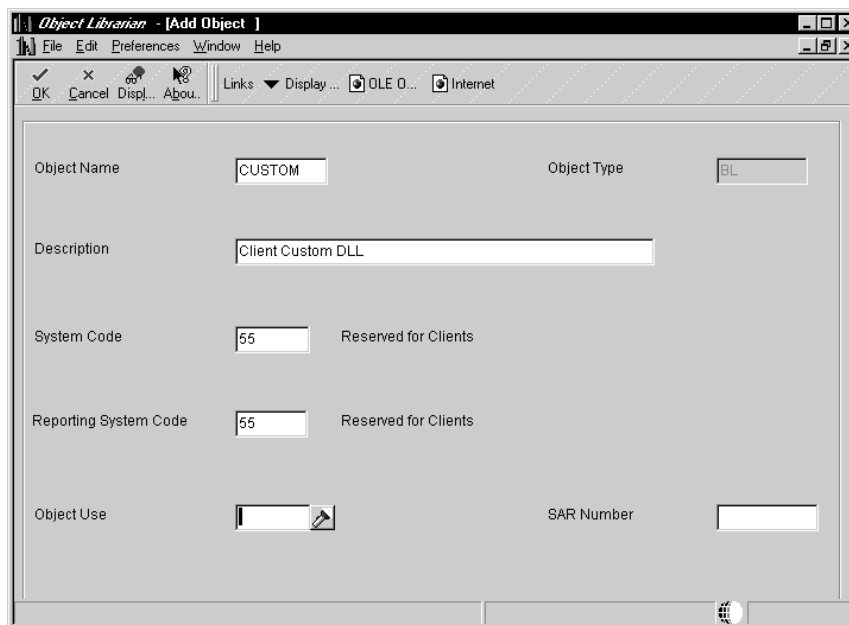
To create a custom DLL

On the Application Development Tools menu (GH901)

1. Choose Cross Application Development Tools (GH902), then choose Object Librarian (P9860).



2. On the Work With Object Librarian form, type BL (for Business Function Library) into the Object Type field, and then click Add.



3. On the Add Object form, enter information into the following fields and click OK:
 - Object Name
 - Description
 - System Code

- Reporting System Code
- Object Use
- SAR Number

Field	Explanation
Object Type	The type of object with which you are working. For OneWorld, on Work with Object Librarian enter an object type and click Add or select an existing one to modify. The Object Librarian automatically executes the specific design tool for the object type. See UDC 98/OF.
Object Name	The name of a unit which consists of attributes and sometimes data. The name of anything that exists in and occupies space in storage, and on which operations can be performed.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
System Code	A user defined code (98/SY) that identifies a J.D. Edwards system.
Reporting System Code	A code that designates the system number for reporting and jargon purposes. See UDC 98/SY.
Object Use	Designates the use of the object. For example, the object may be used to create a program, master file, or transaction journal. See UDC 98/FU.
SAR Number	An abbreviation for Software Action Request (SAR). This number is used by the J.D. Edwards project control system to keep track of all charges and activity by individual SAR number.

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; otherwise, the build process will build the custom business function into the J.D. Edwards CCUSTOM.DLL, which is the default. Use this task to specify a custom DLL whenever you create a custom business function.

To specify a custom DLL for a custom business function

On the Work With Object Librarian form

1. Find, then choose a business function, and click Select.
2. On the Business Function Source Librarian form, check out the business function. Click Design. The Business Function Design form appears.

Function Name	Description	Function Category
CalcTaxAuthTaxAmts	Tax Authority Tax Amounts Calculate	TAX

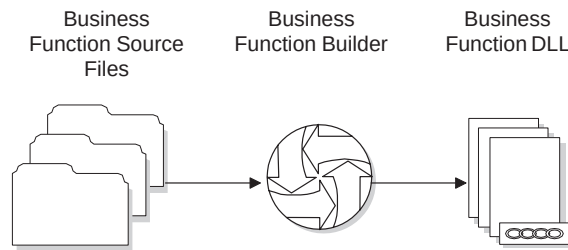
3. Enter your custom DLL name into the Parent DLL field, and click OK. (This same form allows you to change the business function location, if necessary.)
4. On the Work With Object Librarian form, choose that business function again.

The Business Function Source Librarian form appears, and your custom DLL name displays in the Parent DLL field.

5. 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 build business functions includes compiling and linking. Compilation involves creating a business function object. Linking is when you make the object part of a DLL.

The following topics are discussed:

- ☐ Understanding the Business Function Builder form
- ☐ Building a single business function
- ☐ Linking business function objects
- ☐ Setting build options
- ☐ Using utility programs
- ☐ Reading build output
- ☐ Performing a Build All
- ☐ Reviewing error output
- ☐ Resolving errors

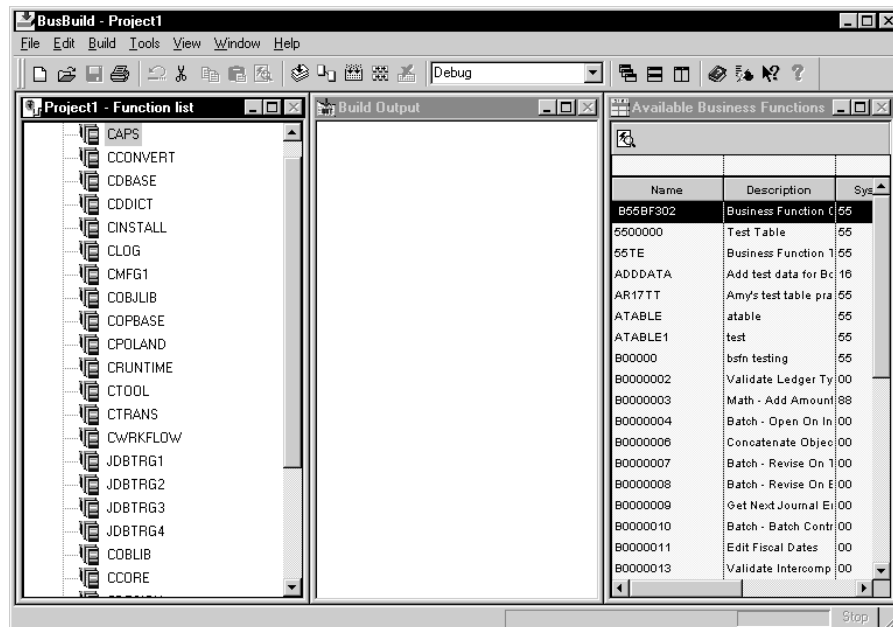
Understanding the Business Function Builder Form

The Business Function Builder form consists of three main components:

- Functions List
- Available Business Functions
- Build Output

In addition to these components, there is also a status window at the bottom of the Business Function Builder form that 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 in the Optimize mode. If you are debugging, the box should be set to Debug mode.



Functions List

Function list displays the consolidated DLLs for OneWorld including custom defined ones. It lists the functions you select to link or compile during a build. Any functions you drag to the function list will build when you build your function list.

Available Business Functions

Available Business Functions displays the functions 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 you can see the build in progress. This display is useful for diagnosing any problems that occur during a build. When the

build is complete, Business Function Builder reports 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, 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. When you create source code changes to a business function you must build the business function to test it.

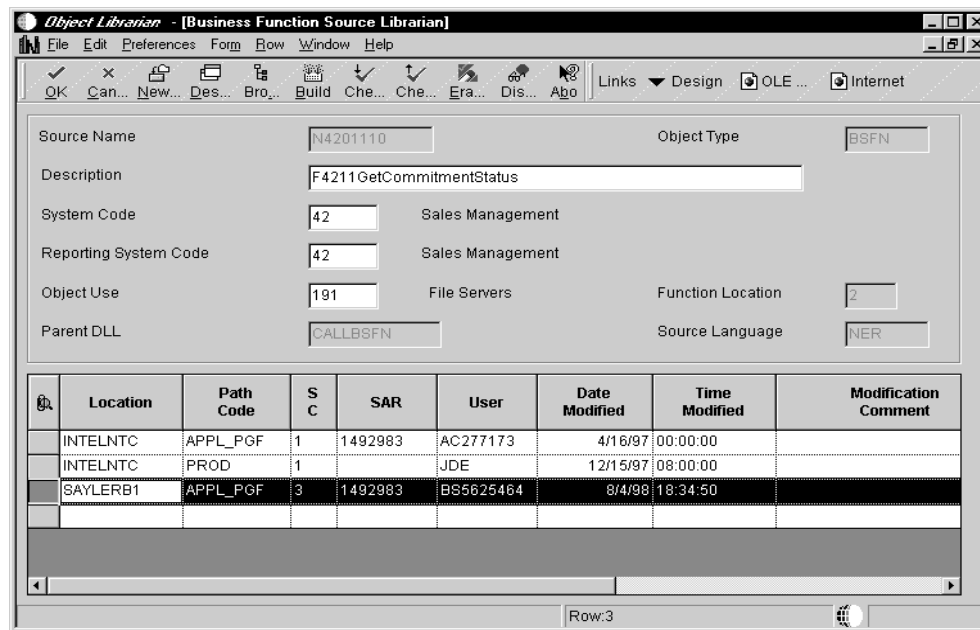
► To build a single business function

On Object Librarian

1. Choose the business function you wish to build.

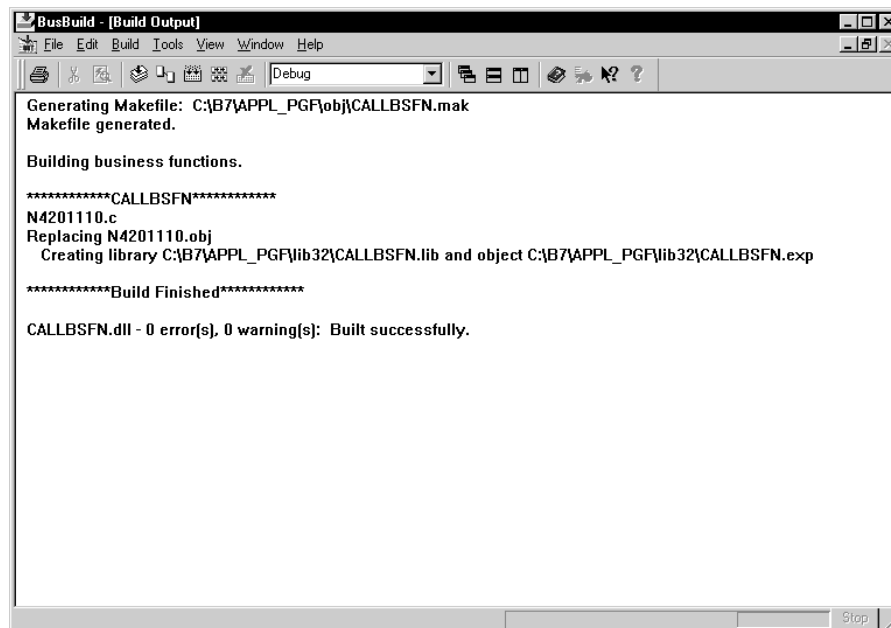
Name	Description	Ins Sys	Rep Sys	Object Type	Object Use	Related Object	Function Type
N4200790	Shipment Confirmation	42	42	BSFN	193		1
N4200860	Backorder Release MBF	42	42	BSFN	131		1
N4200870	Increment Sales Detail Line Number	42	42	BSFN			3
N4200940	Verify and Get Item Master with Short or Unk	42	42	BSFN			3
N4200980	Get Text ID Next Number	42	42	BSFN	330		3
N4201010	Determine If Blanket SO Exists	42	42	BSFN			3
N4201050	Kit Selection in Batch Mode	42	42	BSFN	130		2
N4201060	F42005 Update Commissions	42	42	BSFN	111		3
N4201090	Changed SO Header Values	42	42	BSFN			3
N4201100	F4104 Get XRef Items	42	42	BSFN			2
N4201110	F4211GetCommitmentStatus	42	42	BSFN	191		3
N4201130	Commitments, Kit Balancing	42	42	BSFN	330		2

The Business Function Source Librarian form appears.



- From the Form menu, choose Build.

The BusBuild form appears and Business Function Builder begins building your business function.



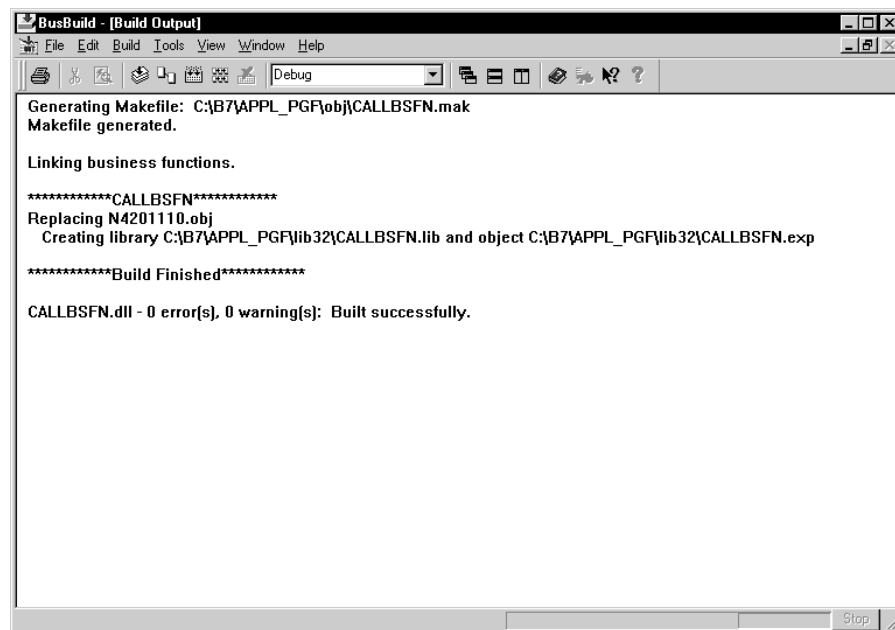
Build Output displays the results of the build. When the build is finished, the message “***Build Finished***” appears at the bottom of Build Output. The text following this line indicates whether the build was successful. If the build was successful you can test your business function. Otherwise, you must correct any problems and try the build again.

To stop the build, click the Stop button in the lower right corner of the form. The DLL will revert back to the original.

Linking Business Function Objects

When you install one or more OneWorld applications on your machine, OneWorld also installs business function objects. OneWorld automatically calls Business Function Builder to perform a Link All operation that integrates these business function objects with your local business function DLLs. This linking preserves your local custom modifications instead of replacing the DLLs. This linking process is not necessary if you do not have custom modifications on your machine.

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.



To set build options

On BusBuild

1. Choose one of the following options from the build menu:

Build	Generates a makefile, compile the selected business functions, and link the functions into the current consolidated DLL. Only components that are out of date will be rebuilt.
Compile	Generates a makefile and compile the selected business functions. The application will <i>not</i> link the functions into the current consolidated DLL.
ANSI Check	Checks the selected business function for ANSI compatibility.
Link	Generates a makefile for each consolidated DLL. It then builds each consolidated DLL. The application does <i>not</i> compile any of the selected business functions.
Link All	Generates a makefile for each consolidated DLL. It then builds each consolidated DLL and links it to all business functions that are called. The application does <i>not</i> 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. Turn this option off to speed up the build.
Precompiled Header	Creates a precompiled header when compiling a business function. When compiling multiple business functions, the Business Function Builder can generally compile 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. Turn this option off to speed up the build.
Full Bind	Resolves all of the external runtime references for each OneWorld consolidated DLL.

Using Utility Programs

The Tool menu contains utility programs that assist in the build process.



To use utility programs

On BusBuild

1. Choose one of the following options from the Tool menu:

Synchronize JDEBLC

You can run this utility to reorganize OneWorld business functions into new DLL groupings. This utility synchronizes the JDEBLC parent DLL field with the parent DLL in the F9860 table. Use this option with caution! You typically use this option only if you have manually dragged business function DLLs from a recent package build and you are experiencing business function load library failures.

Dumpbin

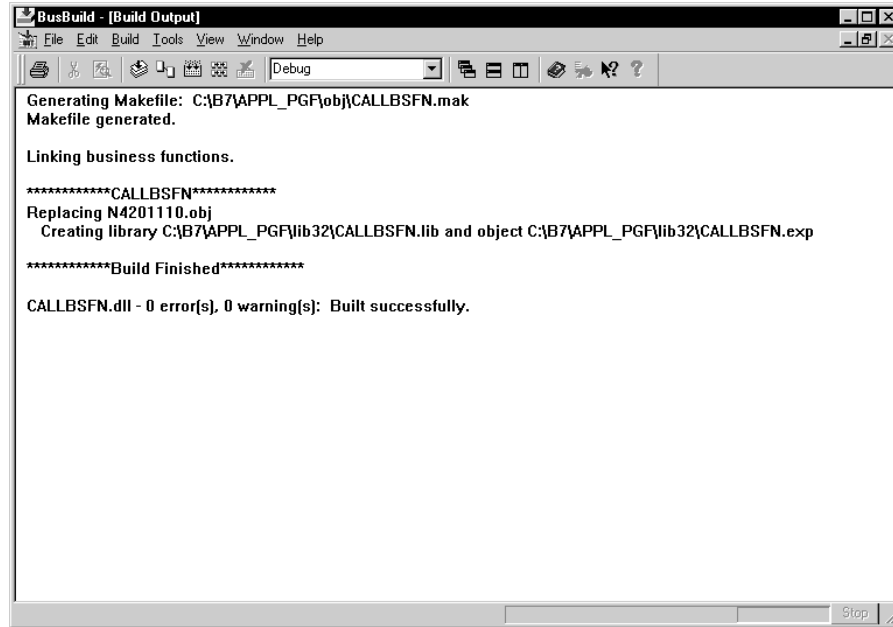
You can run this utility to verify whether a particular business function was successfully built. This utility displays all the business functions that were built into the selected consolidated DLL.

PDB Scan

You will get a CVPACK fatal error if one of the object files 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 output to determine whether your build has completed successfully and to troubleshoot problems if errors occur during your build.



Makefile Section

The makefile section lets you know where Business Function Builder has generated the makefile for this build. Business Function Builder generates one makefile per DLL that it builds. You should always see a Generating Makefile statement for each DLL you are building. If you do not see the makefile statement then something is wrong. To resolve the error:

- Make sure the local object directory exists.
- Make sure 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, the section above begins with *****CDIST*****. You should see a Begin DLL section for each DLL you are building.

Compile Section

In order to build DLLs, Business Function Builder compiles the business functions in the DLLs first. You should see a sequential list of each business function the Business Function Builder attempts to compile. During the compilation process, the following 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. If

you want extended information about the warning you can search on the CXXXX value in Visual C++ helps. Warnings will not usually prevent the business function from compiling successfully. You can turn on the Warnings As Errors option in the Global Build form so that the business function will not build if it has any warnings.

- **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. If you want extended information about the error you can search on the CXXXX value in Visual C++ helps. Errors prevent the business function from compiling successfully and must be resolved.

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 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. If you want extended information about the warning you can search on the LNKXXXX value in Visual C++ helps. Warnings will not usually 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.

- **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. If you want extended information about the error you can search on the LNKXXXX value in 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. It does this by changing the desired load address for the DLL so that the system loader does not have to relocate the image. It 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*****”. Next Business Function Builder displays a summary report for each DLL you attempted to build. This report includes:

- The number of warnings
- The number of errors
- Whether or not the DLL build is successful

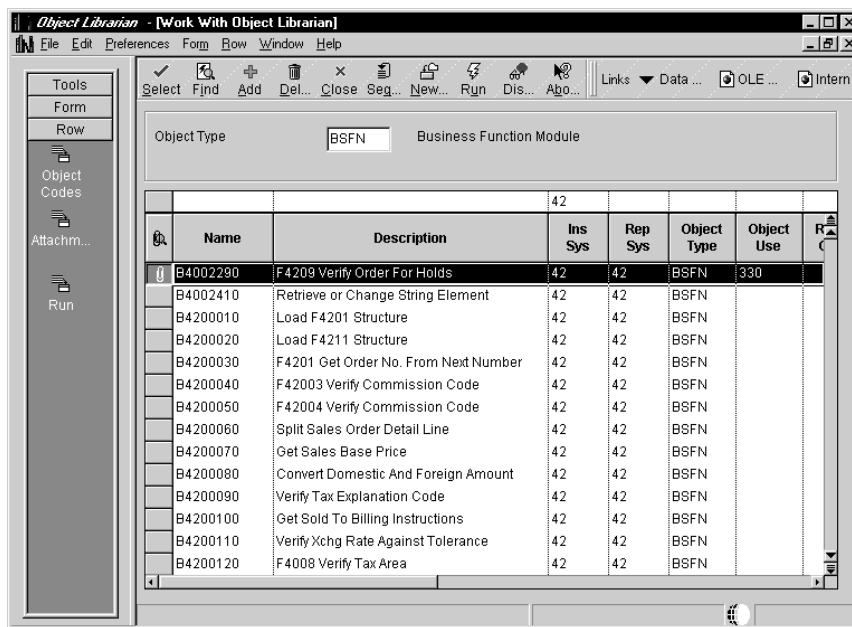
Building All Business Functions

Build All is usually done by a system administrator. When Build All processes it can take a long time. Build All not only does what the global link does but also recompiles all the objects within each DLL. You must access BusBuild from the OneWorld Explorer menu to run Build All.

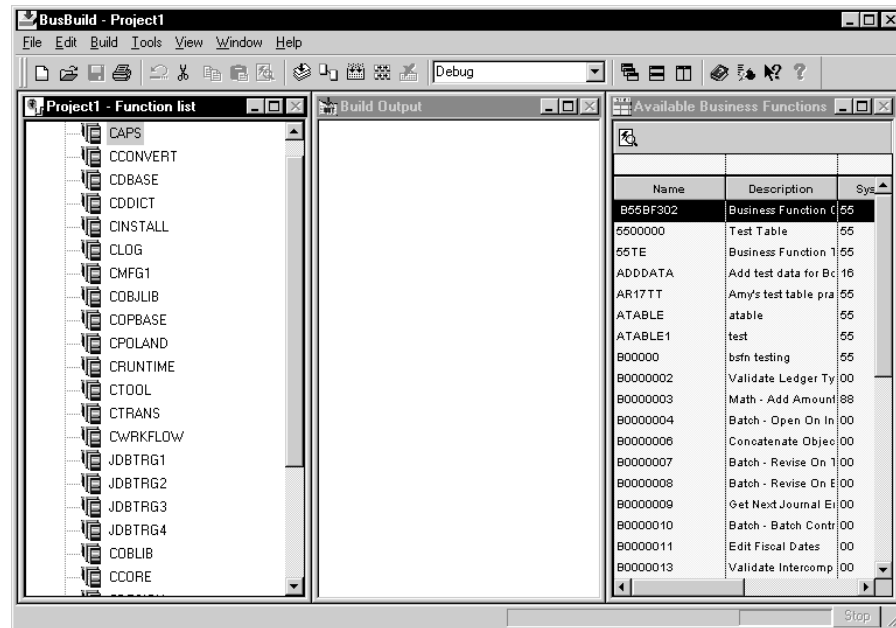
► To build all business functions

On Object Librarian

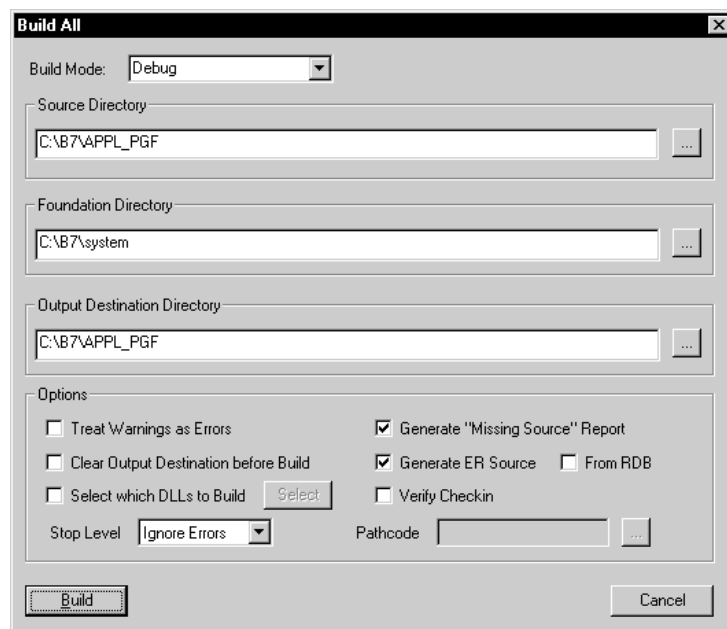
1. From the Form menu, choose Function Builder.



The BusBuild form appears.



- On BusBuild, from the Build menu, choose Build All.



- Choose one of the following options for Build Mode:

Debug

The build will include debug information. After you perform a build, you can debug the built business function using the Visual C debugger.

Optimize	The build will not include debug information. Optimize Builds generally cannot be debugged using the Visual C debugger.
Performance Build	This option should only be used by J.D. Edwards developers. It is the same as an optimize build, except the build includes information to help Tools development measure the performance of business functions.

4. 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. Any full packages usually have all business function source.

Click the button to the right of this field to browse for a directory.

You can choose one of the following locations to indicate where your business function source resides:

Local	All business function source is on the local machine
Path Code	All business function source is in the path specified by the selected pathcode.
Package	All business function source is in the path specified by the selected package. If a package is built correctly, it will typically contain all required business function source. Package is the recommended location.
Pick Directory	All business function source is stored in another directory of your choice that exists on your file server.

For example, suppose that you have a package called PROD_A. PROD_A physically resides on the file server location \\SERVER\SHARE1\PROD_A. 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, you click the button, then Package, then PROD_A from the grid. When Business Function Builder builds, it will use the business function source from this directory.

5. Complete the Foundation Directory field.

Use this field to specify the foundation to use for this build. The foundation you select will be the foundation on which you expect these business functions to run.

Click the button to the right of this field to browse for a location.

You can select your foundation from one of the following locations:

Local The desired foundation is the local OneWorld foundation.

Foundation The foundation table lists all registered OneWorld foundations. Choose a foundation from this table.

Pick Directory The OneWorld foundation exists on your file server in a directory of your choice. This is the recommended choice for this option.

For example, suppose 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

6. Complete the Output Destination Directory field.

Use this field to specify where you want the output of the build to go. The build output includes the following file types: DLL, .LIB, .OBJ, .LOG.

Click the button to the right of this field to browse for a location.

You can select one of the following locations for your output destination directory:

Local	All business function output will go to 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 goes to the selected package. Package is a more stable snapshot of business function source. Package is the recommended location.
Pick Directory	All business function output is stored in a directory of your choice on your file server.

For example, suppose 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 under PROD_A:

Bin32	Contains built .dll files
Lib32	Contains build .lib files
Obj	Contains built .obj files
Work	Contains built .log files

With the above configuration on the server for PROD_A, you click the button, then Package, then choose PROD_A from the grid. When Business Function Builder builds, it places the build output files in these subdirectories for PROD_A.

7. Click any of the following Options:

Treat Warnings As Errors	If you turn on this option, Business Function Builder will not build a business function if it contains any warnings.

Clear Output Destination Before Build	If you turn on this option, Business Function Builder will delete the contents of the bin32, lib32 and obj output directories prior to building all business functions.
Select Which DLLs to Build	If you do not turn on this option, Business Function Builder will build all DLLs. If you turn on this option you can click the Select button to choose which business function DLLs you wish to build. Use this option if you want to build one or two DLLs. If you build only a subset of all DLLs, then you should make sure the Clear Output Destination Before Build option is not turned on.
Stop Level	You can select the error level that the build will stop at. You can ignore errors to continue building despite errors. You can stop if there is a DLL with errors. You can stop on the first compile error.
Generate “Missing Source” Report	This option is recommended. If you turn on this option, Business Function Builder will create a report in the work directory of the destination. This directory is called NoSource.txt. It contains business function source file names that do not have a .c file, but do have an F9860 record. To clean this report you can produce the correct .c file for the business function or you can delete the source file from the F9860 table.
Generate ER Source	If you turn on this option, Business Function Builder will generate named event rule and table event rule source prior to building business functions.
Verify Checkin	If you turn on this option, only objects checked in to a specified pathcode will build. A log file, Notchkdn.txt is written to the same directory as Nosource.txt. Objects not checked into the pathcode will be listed in this log and in Buildlog.txt.

You can turn on the From RDB option so that you can generate work from any path code. If this option is not turned on, the business function builder assumes that the event rules source can be generated from the source directory specification files.

EMPATHCD	EMDL01	EMMKEY	EMSERSI
APPL_PG	Application Deve	INTELNTC	OWENVIRO
B7141	Cumm B71.4.1	INTELNTC	OWENVIRO
B7142	Cumm B71.4.2	INTELNTC	OWENVIRO
B7311LANG	MASTERS B731	INTELNTA	ONEWORLD
B7312	Pristine B73.1.2	INTELNTC	OWENVIRO
B7313	DO NOT CHECK	INTELNTC	OWENVIRO
B731LANG	MASTERS B731	INTELNTA	ONEWORLD
BLDCTL	Build 21 control t	BUILD21	b7

Grid Rows: 40

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, you simply click Build to rebuild the problem DLLs.

Field	Explanation
Build Mode	This should usually be in the Optimize mode.
Source Directory	This is the path code Business Function Builder will retrieve the source from, in case of multiple path codes on a client machine. This is the path to the .c and .h files.
Foundation Directory	This designates which system the build is built on. This can either be your local client system, or a system can be used from the server.
Destination Directory	Where the output of the build will be stored.
Treat Warnings as Errors	When building business functions you have the option of reporting all compiler warnings as errors or as warnings. The consolidated DLL for a function with a warning or error will still be built. However, if the function has an error, that function will not be built into the consolidated DLL. Functions with warnings will be built into the consolidated DLL.
Clear Output Destination before Build	When building business functions, you should normally clear the output directories (bin32, lib32 and obj) so that old business functions are not included in the build. If you do not clear the output and a function contains errors, the global link process will use the old libraries and objs to make the consolidated DLL. This may not give you an accurate global build.

Field	Explanation
Select which DLLs to Build	When doing a global build, you can choose individual DLLs to be built.
Generate Event Rule Source	This option usually comes turned on, but for performance reasons can be turned off. It is used to generate event rule source such as event rules attached to tables (TER) and business functions written in “NER”. The source for these is usually generated when these objects are created, making it unnecessary to generate the source again.
Generate Missing Source Report	This option shows all source members that were not present at the time of the business function build. The build attempted to find these members because each had a record in the Object Librarian Table (F9860). However, a matching source could not be found in the “source” directory. To resolve these errors either delete the Object Librarian record or provide a source member.

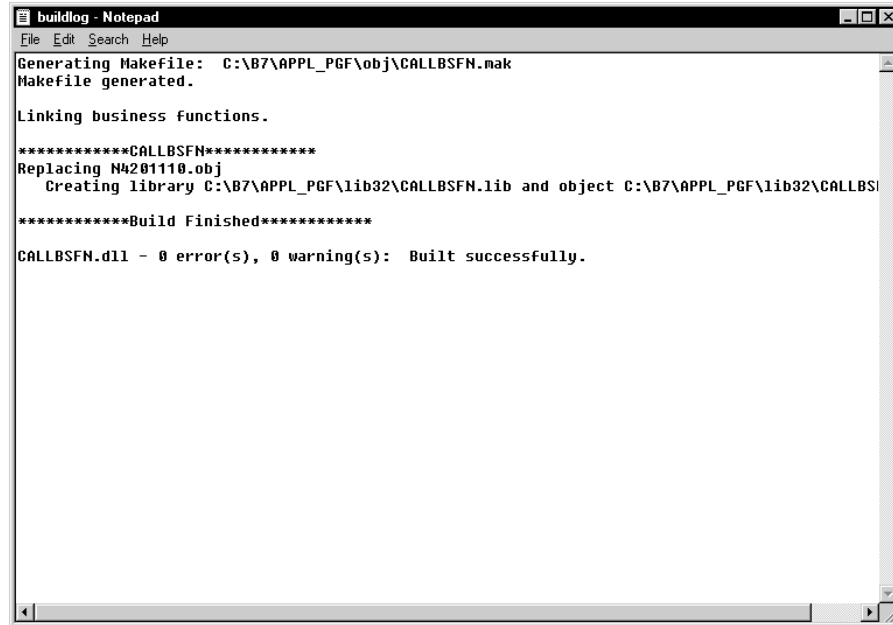
You can also run this build by turning the Build BSFN option on in a package build. Refer to the *Package Management Guide* for more information about this option.

Reviewing Error Output

A “work” directory is created when any object is built. This directory is in the destination directory you have selected, for example: 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.

A sample log is shown below:



```
buildlog - Notepad
File Edit Search Help
Generating Makefile: C:\B7\APPL_PGF\obj\CALLBSFN.mak
Makefile generated.

Linking business functions.

*****CALLBSFN*****
Replacing N4201110.obj
Creating library C:\B7\APPL_PGF\lib32\CALLBSFN.lib and object C:\B7\APPL_PGF\lib32\CALLBSFN.lib

*****Build Finished*****

CALLBSFN.dll - 0 error(s), 0 warning(s): Built successfully.
```

Resolving Errors

You can use Business Function Builder tools to help you resolve problems. If you notice any unresolved external errors during a business function build your consolidated DLL will still build and OneWorld should run normally. However, OneWorld will not be able to execute any business function that was not resolved.

Use the dumpbin tool to verify that a particular business function is present in a consolidated DLL. If a business function is present, you will find its name 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 information. The PDB scan will find 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 they work properly.

If one of the DLLs gets out of synch, you must rebuild it using the Build option. This will generate a makefile and then relink all the business functions within it.

The Synchronize JDEBLC option from the Business Function Builder Tools menu corrects the problem of any misplaced or incorrectly built business functions. This option will look at the server DLLs and determine 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	The build header shows the configuration for a specific build, including the source path, foundation path, and destination path.
Build Messages	The build messages section displays the compile and link activity. During a compile, a line is output for each business function that was compiled. Any compile errors will be reported as “error cxxx”. During the link part, business function builder outputs the text “Creating library”. This text may 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: “x error(s), x warnings (y)”. The summary indicates how the build went. If you have no warnings and no errors, then the build was successful. If the summary reports an error, you should search the log for the word “error” 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 defaulting and editing for a transaction file. Records are dependent on each other. They contain logic that ensures the integrity of the transaction being inserted, updated, or deleted from the database. Logic is broken up by event flow. You use Cache APIs to hold records being processed. You should consider using a transaction master business function when:

- You accept transaction file records from a non-J. D. Edwards source
- Multiple applications are updating the same transaction file

For example:

- The Account Ledger table (F0911) accepts updates across product lines as well as external sources.
- The Time Entry table (F06116) accepts updates from batch, interactive and external sources.

The following topics are discussed:

- ☐ Creating processing options for a master business function
- ☐ Naming Transaction Master Business Function modules
- ☐ Implementing Transaction Master Business Functions

Master Business Function Naming Convention

The Master Business Function source is named Bxxxxxxx (the same as any other business function), unless the object already exists on the AS/400 as a functional server. In this case the XTXXXXZ1 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. (for example 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*. Document will be shortened to Doc. (for example, F0411BeginDoc)

The functional use code of 192 in the Object Librarian is used to designate business functions as master business functions.

A cache is used to keep track of transaction records. It has the following naming convention:

FxxUIyyy

xx is system code

yyy is unique number

Creating Processing Options for a Master Business Function

Because Master Business Functions (MBF) can be called from several different applications, rather than duplicating processing options needed by the master business function on each application, these processing options should reside in their own processing option template.



To create processing options for a Master Business Function

1. Create a processing option template. The processing option template name format is Txxxxxxx (replace the B or XT in the MBF name with a T).
2. Create a dummy application to attach the MBF processing options to. Different versions of the MBF processing options can then be set up through versions list. The naming convention for this dummy application is Pxxxxxxx (replace the B or XT in the MBF name with a P). This application needs only one form with no fields selected in order to generate.
3. You can set up different versions of the master business function processing options using interactive versions. Calling programs then simply pass in the version name in the version parameter of BeginDoc.
4. 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.

Refer to the online APIs for more information.

Naming Transaction Master Business Function Modules

Function names follow the standard naming convention Fxxxxx. If a table has multiple master business functions, you should put the program name in the function name.

FxxxxxProgram Name Module

For example:

- F3112SuperBackFlushBegDoc
- F3112WorkOrderRoutingsBegDoc

Components of a Transaction Master Business Function

There are several basic components that are typically used for a master business function.

Begin Doc	Called when all header information has been entered • Creates initial header if not already created. May also include defaulting, editing, and processing options.
Edit Line	Called when all line information has been entered • Creates cache for detail information if not already created.
Edit Doc	Called when ready to commit the transaction • Processes any last document edits and verifies that all records are valid to commit.
End Doc	Called when you need to commit the transaction • Processes all records in the header and detail cache, and 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?

- Header level defaulting and editing. This includes data dictionary defaults, edits, and UDC editing.
- Fetch to the database, if necessary, to validate that the selected document action can take place.

- Any validation or processing that is common to all records.
- Write record to header cache if there are no errors.
- Header cache will contain all information that is common to all detail records.
- This improves performance because all the detail records are not needed to do the same validations and table I/O.
- If there is a change to the header fields and Begin Document has been called previously, then the program updates the header cache with the new information.

Special Logic/Processing Required

- On the initial call, the job number will be assigned by the function. This function will call X0010GetNextNumber with a system code of “00” and a Index Number of “04” to get the job number. If Begin Document is called again, the job number previously assigned will be passed; therefore, there is no need to assign another job number.

Hook-Up Tips

- Call a function at least once before calling Edit Line
- If there are errors during validation of the header field when the function is called, the function might need to be called again to make sure errors have been cleared before calling Edit Line
- If this function might be called multiple times from different events it might be a good idea to put 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 detail records or deleting and is finished adding header information.
- This button might then be called on OK button in case of a Copy where the user will not touch the grid.
- 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; else pass zeros and this function will assign one.

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: GUI interface will usually use the partial edit, and the batch interface will use the full. If blank, defaults to Full edits.
ErrorConditions	EV02	O	" " = 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 will process all these fields and validate them, Data Dictionary edits, and UDC editing, defaults and so on. Begin document may also fetch to the file 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	
Name	Alias	I/O	Description
Work Field / Processing Flag One	*****	I	List any work fields that are needed by the Program. These could be flags for processing, dates to validate, and so on. These fields may or may not be used. An example would be currency control. The user would want this 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 needed to process header level information
- List any work fields needed to perform edits
- List all processing options needed to process header level information

Edit Line

Function Name

FxxxxxEditLine

What does it do?

- Validates all user input, perform calculations, and to retrieve default information. Edit Line is normally called for every record that is fetched and performs the edits for that *one* record in the file.
- Reads header cache records for default values.
- On an ADD, defaults information into blank columns, such as Address Book information. This default value may be coming from:
 - 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, UDC and data dictionary edits are performed.
- Writes record to the detail cache if error free. 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 record's action code will be changed to delete, and the record will be stored in the detail cache until file processing in End Doc.

Special Logic/Processing Required

- Depending on the type of document being processed, different editing and defaulting will take place. An example would be vouchers and invoices processed through the journal entry MBF. The tax calculator is only called for Journal Entries.

- Depending on the event processing required, the process edit flag will determine which editing will be performed. For example, in an interactive program, when the *Grid Record is Fetch* event runs, Partial Edits may be performed to retrieve descriptions, default values, and so on. When the *Row is Exited and Changed* event runs, Full Edits may 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	Retrieved from Begin Document. Used as key or to create an unique name for the Cache/Work file
Line Number	LNID	I/O	The unique number identifying the transaction line. Can also be used as the line number into 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 will most always use the partial edit and the batch interface will use the full. If blank, defaults to Full edits
Error Conditions	ERRC	O	' ' = No Errors '1' = Warning '2' = Error
Update Or Write to Work File	EV02	I	'1' = Write and/or Update records to the work file
Record Written to Work File	EV03	I/O	Will be set to a '1' if a record is written to the work file. This will save I/O calls to the work file. Will be ' ' if no record was 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 will provide validation, Data Dictionary edits, and UDC editing, defaults etc.
Detail Field Two	*****	I/O	
.	*****	I/O	
.			
.			

Detail Field XX	****	I/O	
Work Field / Processing Flag One	****	I	List any work fields that are needed by the Program, this could be flags for processing, or dates to validate etc.
Work Field / Processing Flag One	****	I	
.		I	
.			
.			
Work Field / Processing Flag One		I	

Edit Document

Function Name

FxxxxxEditDocument

What does it do?

- Reads cache records if multiple line editing is required.
- Read header cache record if header information is needed.
- Performs cross dependency edits involving multiple lines in document.
- An example would be processing all records to ensure that percentages total 100%.
- An example would be validating that the last record can't contain certain information.

Special Logic/Processing Required

- Depending on the type of document being processed different logic will be executed. An example would be vouchers and invoices processed through the journal entry edit object. The balancing is different for these document types.

Hook Up Tips

- You should call the function at least once after calling Edit Line and before End Document
- If there are errors during validation, the function may need to be called again to make sure errors have been cleared before calling End Document.
- This function should be called on the OK *button clicked* event so that if there are any errors, they get fixed before the user exits the application.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	Was retrieved from Begin Document
ErrorConditions	EV01	O	' ' = 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, a minimum number of parameters are needed in this function.

End Document

Function Name

FxxxxxEndDocument

What does it do?

- Assigns next number to document. For vouchers, you should do this before calling journal entry edit object, but not before voucher has been balanced and is truly going to be committed to the data base. By placing this module on the *before add/delete/update* event, the document will have passed all edits before running this.
- 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 and updates associated tables. For example:
 - Manual checks associated to Vouchers
 - Address Book vouchered YTD columns in Address Book
 - Address, phones, and who's who for Address Book.
 - Batch header
- After all updates have successfully completed, cleans up the cache for that document and any work fields.
- Summarizes documents, if designated in a processing option, as it is writing to the database. Will read work file through an alternate means and write the records at a control break.
- Currency Conversion.

Special Logic/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	Was retrieved from Begin Document
Computer ID	CTID	I	Retrieved from GetAuditInfo(B9800100) in application (optional)
Error Conditions	EV01	O	‘ ‘ = No Errors ‘1’ = Warning ‘2’ = Error
Program ID	PID	I	Usually hard coded

Application Specific Parameters

- List the fields needed to process update or writes, for example Time and Date Stamp fields
- List any work fields needed to perform updates or writes
- List all processing options needed to process updates or writes

Clear Cache**Function Name**

FxxxxxClearCache

What does it do?

- Removes the records from the header and detail cache

Special Logic/Processing Required

- If a unique cache name is selected as the naming convention for the cache (Dxxxxxxx [Job Number]), then you would use the cache API `jdeCacheTerminateAll` to destroy the cache.

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	The job number of the transaction that you wish to clear. This job number should have been returned from BeginDoc.
Clear Header	EV01	I	Indicates if the header cache should be cleared '1' = clear cache
Clear Detail	EV02	I	Indicates if the detail cache should be cleared '1' = clear cache
Line Number From (Optional)	LNID	I	The line number indicating where to begin clearing records in the detail cache. If this is blank, will begin clearing from the first record.
Line Number Thru (Optional)	NLIN	I	The line number indicating where to stop clearing records in the detail cache. If this is blank will delete to the end of the cache.

Cancel Document (optional)**Function Name**

FxxxxxCancelDoc

What does it do?

- Application specific. Used primarily with the Cancel button to close files, clear the cache, and so on. Provides basic function cleanup.

Special Logic/Processing Required

- Application Specific

Common Parameters

Name	Alias	I/O	Description
Job Number	JOBS	I	The job number of the transaction that you wish to clear. This job number should have been returned from BeginDoc.

Cache Structure

The cache structure is used to store 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:

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

This is a unique job number assigned when the job is started by the BeginDoc module. This will distinguish 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 could therefore use any field as the key to the cache.

Document Action

The action for the document
A or 1 = Add
C or 2 = Change
D = Delete

Processing Options

Processing option values, which were read in using AllocatePOVersionData, that are needed in other modules of the MBF.

Currency Flag

System value, which tells you if currency is on and what method of currency conversion is used (N, Y or Z).

Business View Fields

These are the fields required for processing the transaction, which will be written to the database. All fields in the record format that are not saved in the header cache will be initialized when record is added to the database using the APIs.

Work Fields

These are fields that are not part of the business view but are needed for editing and updating the transaction.

For example, Last Line Number:. This 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 back to the header cache and stored for the next transaction.

Detail

Field Description	Field	Key	Type	Size
Job Number	JOB5	X	Char	8
Line Number	(Application Specific)	X	Num	
Line Action	ACTN		Char	1
Business View Fields				
Work Fields				

Job Number

This is a unique job number assigned when the job is started by the BeginDoc module. This will distinguish transactions in the cache for each job on the client that is using the cache. 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 could therefore use line number only as the key to the cache.

Line Number

This will need to be the line number used to uniquely identify lines in the detail cache. This line number could 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
 A or 1 = Add
 C or 2 = Change
 D = Delete

Business View Fields

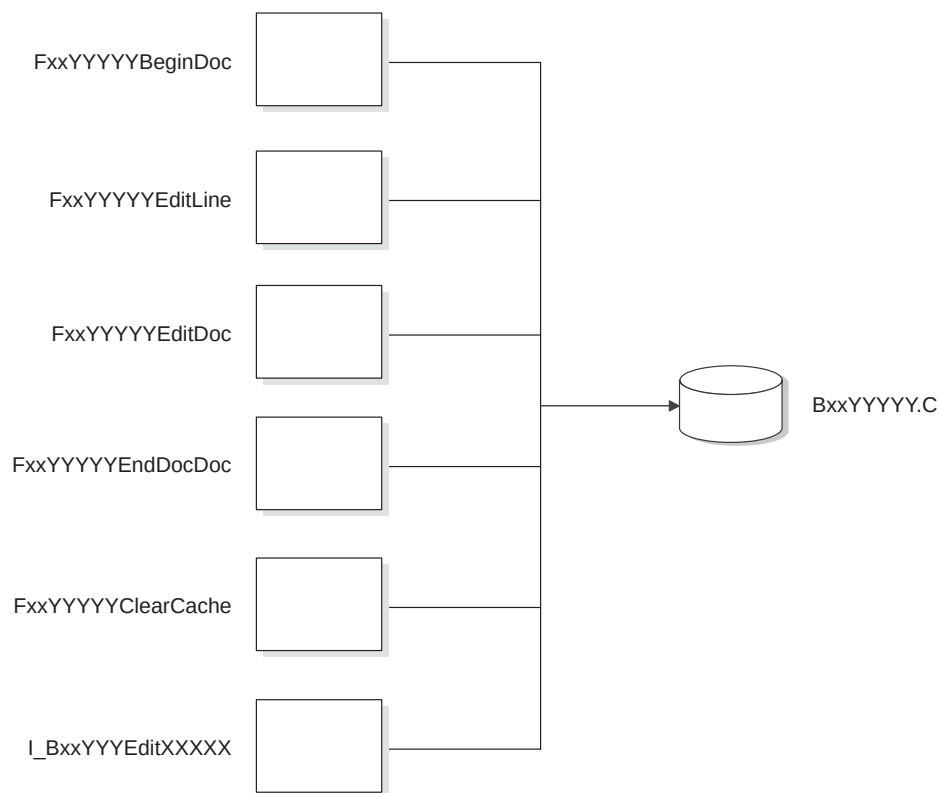
These are the fields required for processing the transaction which will be written to the database. All fields in the record format which are not saved in the detail cache will be initialized when record is added to database using the API's.

Work Fields

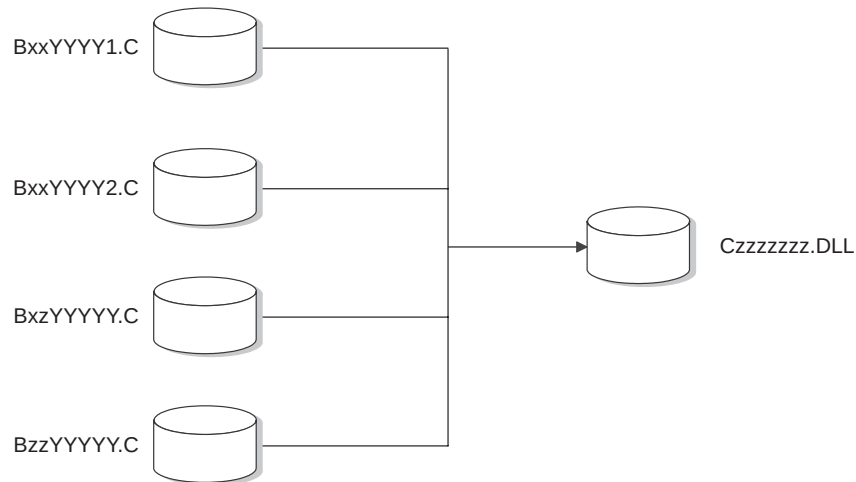
These are fields that are not part of the business view but are needed for editing and updating the transaction line.

Building Transaction Master Business Functions

The following illustrations show how transaction master business functions are built. First you create the individual business functions using the components described above.



Next you combine the 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

1. *Post Dialog is Initialized* (optional)
 - Hook up Begin Document function
2. *Set Focus on Grid*
2. *Row is Exited and Changed* or *Row is Exited and Changed ASYNC*
 - Hook up Edit Line function
3. *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
4. *OK button pressed event*
 - Call Begin Doc
 - Hook up Edit Document function
5. *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.

1. *Add Grid Record to DB - before*
 - Suppress Add
2. *Update Grid Record to DB - before*
 - Suppress Update
3. *Delete Grid Record to DB - before*
 - Suppress Delete

Batch Program Flow Example

1. Do Section of Report Header
 - Hook up Begin Document function
2. Do Section of the Group Section
 - Hook up Edit Line function
3. Do Section of a Conditional Section (optional)
 - Hook up Edit Document function
4. Do Section of Report Footer
 - Hook up End Document function

Document Processing

Program Flow Example

1. *Dialog is Initialized*
 - Hook up Open Batch Edit Object module
2. *Grid is entered* - Finished entering in header
 - Hook up Begin Document Edit Object module
3. *Row is exited* -
 - Hook up Edit Line Edit Object module
4. *OK button is pressed*
 - Hook up Edit Document Edit Object module
5. *Before Add/Delete from Database event*
 - Suppress Add/Delete
 - Hook up End Document Edit Object module
6. *Cancel button is pressed*
 - Hook up Close Batch Edit Object module

Master File Master Business Functions

J.D. Edwards provides master business functions (MBF) to allow calling programs to process certain predefined transactions. A MBF encapsulates the required logic, enforces data integrity, and insulates the calling programs from the database structures.

MBFs are similar to the Functional Servers in World software. You can use MBFs for the following reasons:

- Application specific code is reuseable
- Less duplicated code
- Hook up is consistent
- Supports interoperability models
- Processing can be distributed through OCM
- Design for 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 the need for maintenance from external parties. The requirements for maintaining master files are different than for multiline business transactions.

Address Book and Item Master require MBF support.

Generally speaking, master file MBFs are much simpler than multi-line business transaction MBFs. Transaction MBFs are specific to a program, while master file MBFs perform multiple hits on a table.

Master file MBFs can be used instead of table I/O for interoperability. 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 should be able to make one call to an MBF to edit and add, update, or delete a master file record. An edit-only option should also be available.

Single data structure	The fields required to make the request and provide all the necessary values should be 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, there is no need to complicate matters with the use of caching. The information provided with each call and the current condition of the database should always give the MBF all the necessary information it needs to perform the requested function.
Normal error handling	As with other MBFs they must be capable of executing in both an interactive and batch environment. Therefore, it must be left to the calling program to determine the delivery mechanism of the errors.
Inquiry feature	To allow external systems to be totally insulated from the J.D. Edwards 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.
Impact on applications	For OneWorld applications the impact of implementing a master file MBF should be minimal. There are several standards that should be considered and followed before master file MBF implementation.

Master file applications use the OneWorld tool to process all I/O for the “Work With” (or find/browse) forms. This allows you to use all of the search capabilities of OneWorld.

You should design all master file applications so that all Fix/Inspect forms are independent of each other. This implies that each Fix/Inspect form will use the OneWorld tool to fetch the record, and all edits and updates will happen using the master file MBF. This independent design has two major benefits:

- Organizing the application in this manner simplifies issues related to edits involving dependent fields across multiple forms.
- This design allows consistent implementation of “modeless processing” across Master File applications and all forms within these applications.

There are certain circumstances that may justify deviation from this simple model. These are:

- Extremely large file formats

In the event that 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 may have to be split. The suggested technique is to split the MBF into one that handles “base” data and performs all adds and deletes, and one or more others that allow the calling program to update additional data when the base data has been established. When this is the case, it is usually logical to split it regardless of the technical limitation.

For example if the customer master file exceeded the data structure limitation, you would use two MBFs to process the file:

- F0301ProcessMasterData
- F0301ProcessBillingData

In this example the F0301ProcessMasterData function would process the base data and the F0301ProcessBillingData would be used to update additional data.

- Subordinate detail files

There may be information in addition to the primary master file that has been normalized out to allow for a one-to-many relationship. If we designed our Master File MBF strictly on the basis of how the database is designed, that would translate into three calls. It is valid to include in the data structure of a Master File MBF at least one occurrence of a detail relationship. This will allow users to establish reasonably complete master file information using a simple interface as long as they have simple needs.

Street addresses and phone numbers within Address Book are a good example. It is reasonable for our customers to expect that they could call a simple address book API with basic identifying information, the street address, and a phone number to create an address book record.

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
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 would be optional. All other Action codes, this would be required.
Same as except	AN8	I	Optional	Required for S - Action Code

Application Parameters (Example: Address Book)

Name	Alias	I/O	Required/ Optional	Description
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	ADDS	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, the same as 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 the x's are replaced with 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.

Performance Impact

Regardless of how you handle large format tables, there may be performance implications. Two options are:

- Logical groupings of data allow data structures to be smaller and easier for the user to implement. However, this forces the user to make multiple calls to add or update an entire record in a table.
- You can use a data structure that allows 300 fields. This is also cumbersome to implement and the user may choose not to apply all of the fields.

Through different interfaces, the user may add additional data at a later time anyway. Most processes dictate that part of the data be added now while other related data is added later. For example, the user may define a customer master, yet wait to define the customer's billing instructions until a later date. For this reason alone, the first option of splitting MBFs into one MBF that handles base data and one MBF that handles additional data is suggested.

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:

- Purpose
- Parameters (the data structure used)
- Explanation of each individual parameter that indicates: input/output required and 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.

Business function documentation features allow you to:

- Create documentation
- Generate documentation
- View documentation

Creating Business Function Documentation

You can create business function documentation at several levels, including:

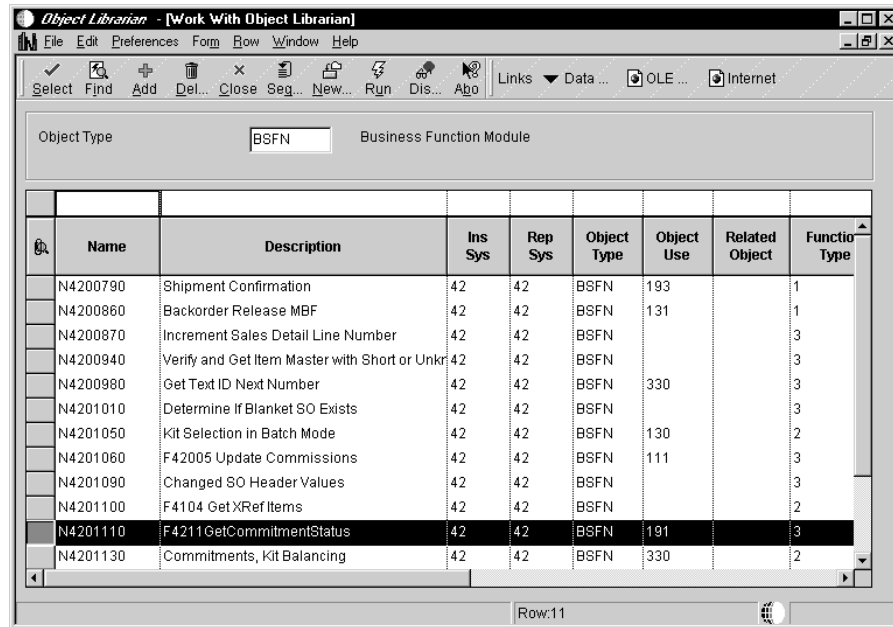
Business Function Notes	This shows you the documentation for the specific business function you are using.
Data Structure Notes	This displays notes on the data structure for the business function.
Parameter Notes	This displays notes on the actual parameters in the data structure.

Creating Business Function Documentation

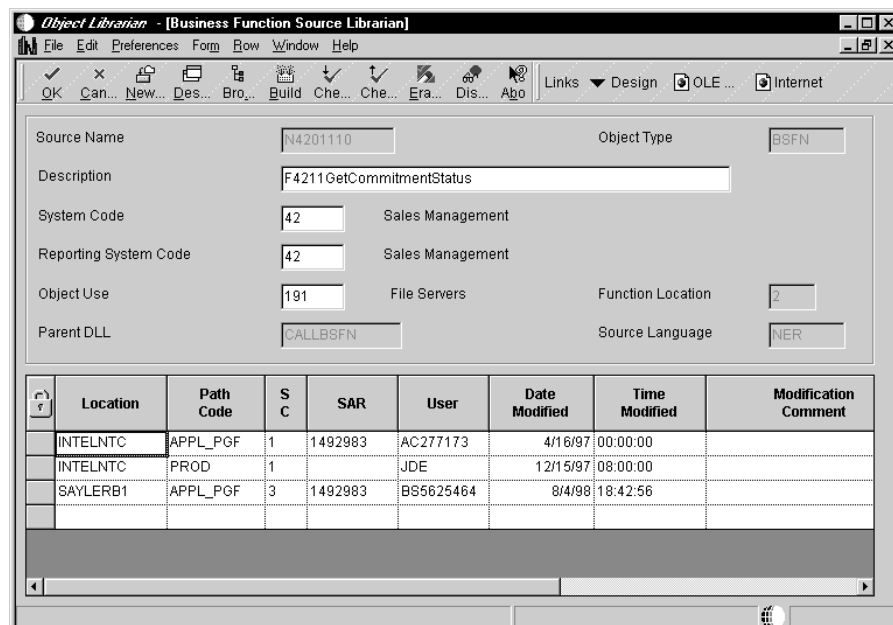
► To create business function documentation

On Object Librarian

1. Choose the business function you wish to document.



2. Check the business function out to your workstation.



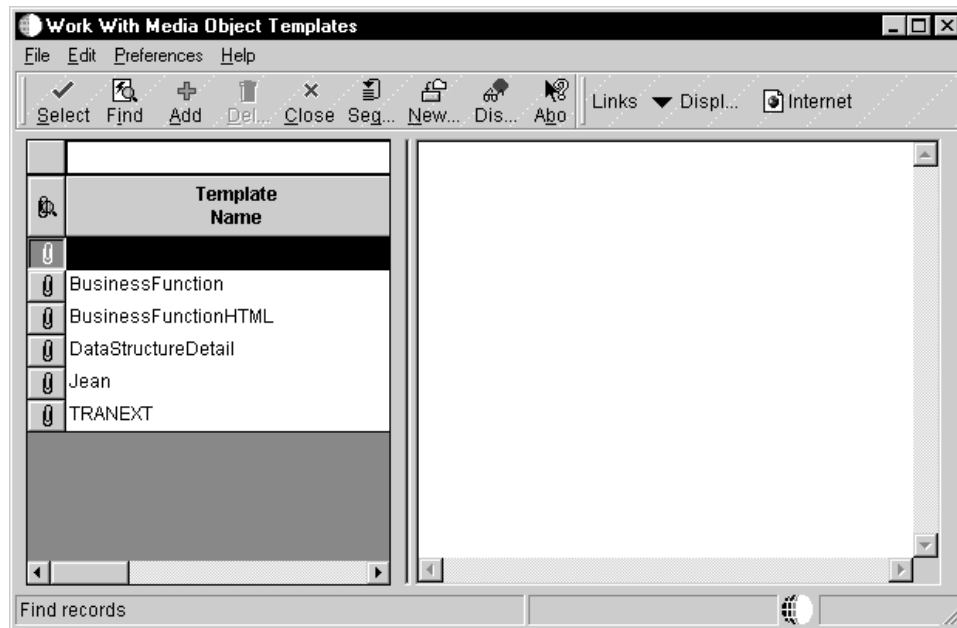
3. From the Form menu, choose Design.

Function Name	Description	Function Category	F
F4211GetCommitmentStatus	F4211 Get Commitment Status	GET	

4. Choose the function you wish to document.
5. Choose Attachments from the row menu.

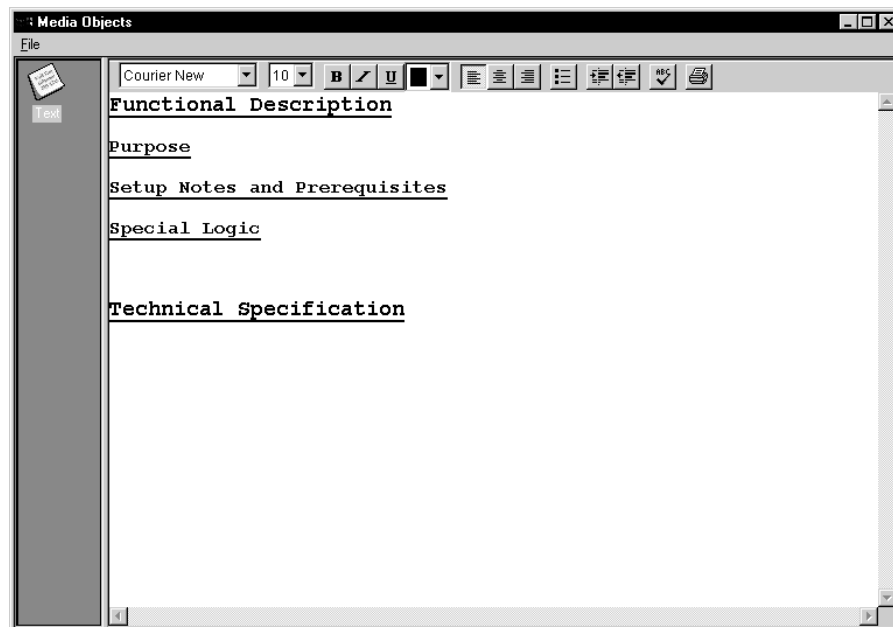
6. On Media Objects, right click in the icon panel and choose Templates.

Work with Media Object Templates displays the available templates you can use.



7. Select the template you wish to use.

J.D. Edwards uses the Business Function template as a standard.



8. Type in the appropriate information under each template heading.

Business Function Documentation Template

The Business Function documentation template contains the following sections:

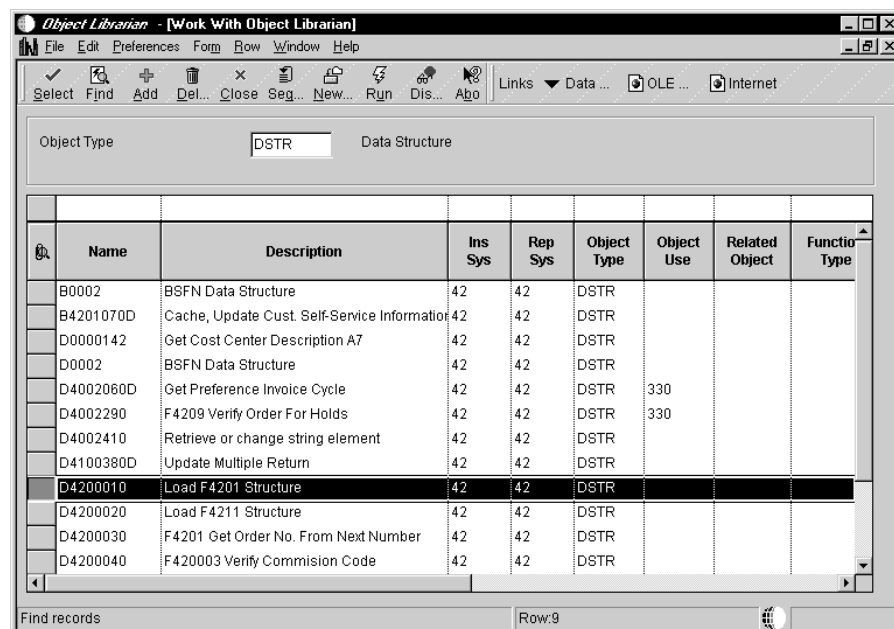
Purpose	This section contains a brief summary of what the function does.
Setup Notes and Prerequisites	The section includes any special notes to assist in using the function, including prerequisite functions, special values that need to be initialized, events recommended to run the function, or if memory must be cleared separately after the function is used.
Special Logic	This section contains additional details about the business function logic. It is usually only used 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. This may be a direct copy from an existing word processing document.

Creating Data Structure Documentation

► To create data structure documentation

On Object Librarian

1. Choose the data structure you wish to document.



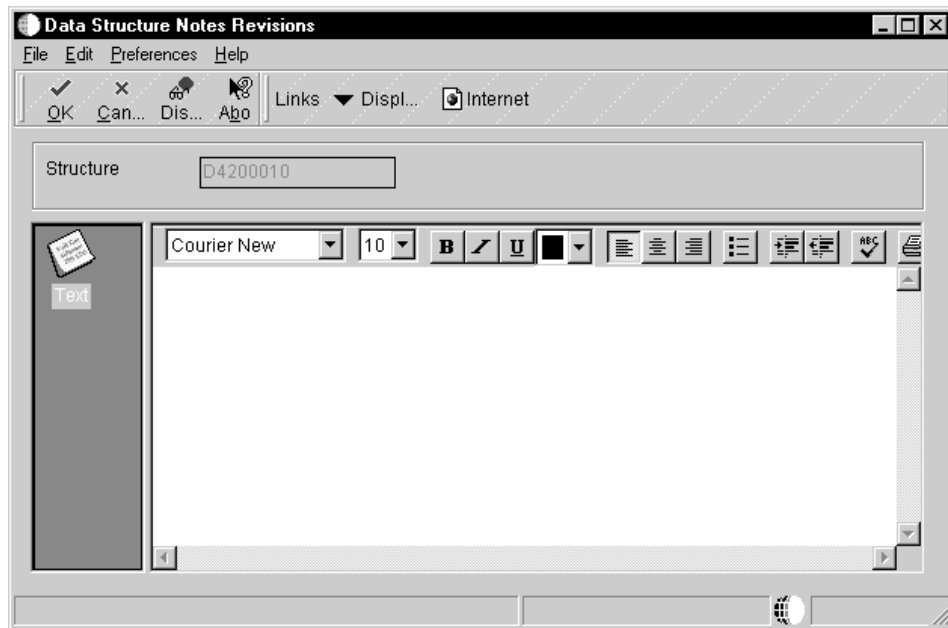
2. Check the data structure out to your workstation.

Location	Path Code	S C	SAR	User ID	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PG	1		SAI VAKKAL	3/2/95	18:24:49	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PG	3		BS5625464	8/4/98	18:46:10	

- On Parameter Design, choose Design from the Form menu.

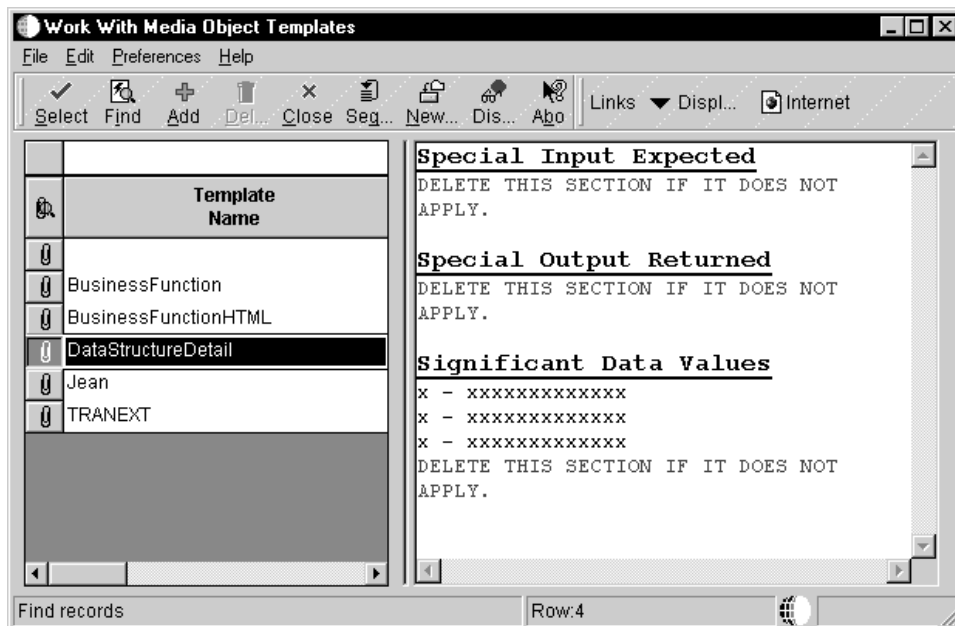
Name	Data Item	Req	I/
szOrderNumberDocumentComp	CompanyKeyOrderNo		
mnOrderNumber	DocumentOrderInvoiceE		
szOrderType	OrderType		
szOrderSuffix	OrderSuffix		
szBusinessUnit	CostCenter		
szCompany	Company		
szOriginalOrderDocumentComp	CompanyKeyOriginal		
szOriginalOrderNumber	OriginalPoSoNumber		
szOriginalOrderType	OriginalOrderType		
szRelatedOrderKeyCompany	CompanyKeyRelated		
szRelatedPoSoWoNumber	RelatedPoSoNumber		
szRelatedPoSoWoOrderType	RelatedOrderType		
mnAddressNumber	AddressNumber		

- On Data Structure Design, click the left binder clip for Data Structure Attachments.



5. On Media Objects, right click in the icon panel and choose templates.

Work with Media Object Templates displays the available templates you can use.



6. Select the template you wish to use.

J. D. Edwards uses the Data Structure Detail template.

7. Type in the appropriate information under each template heading.

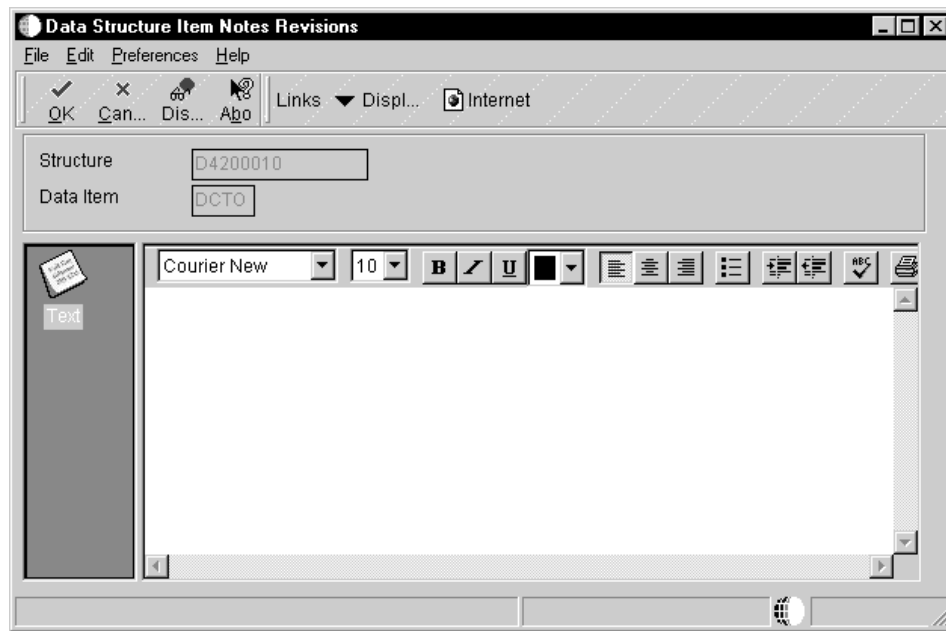
Data Structure Documentation Template

The data structure documentation template contains the following sections:

Special Input Expected	This section should be deleted if it does not apply.
Special Output Returned	This section should be deleted if it does not apply.
Significant Data Values	This section should be deleted if it does not apply. Otherwise it is in the format: x - xxxxxxxxxxxxxx.

Creating Parameter Documentation

The steps for creating parameter documentation are the same as those for data structure documentation, except that after selecting the data item in the structure you wish to enter notes for, you click the Data Structure Item Attachments binder clip instead of the Data Structure Attachments binder clip. There is no special template for parameter documentation.



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, for example a mode parameter. The notes should indicate the valid values the function will except when you hook it up and how to use them. For example, 1 = Add mode, or 2 = Delete.

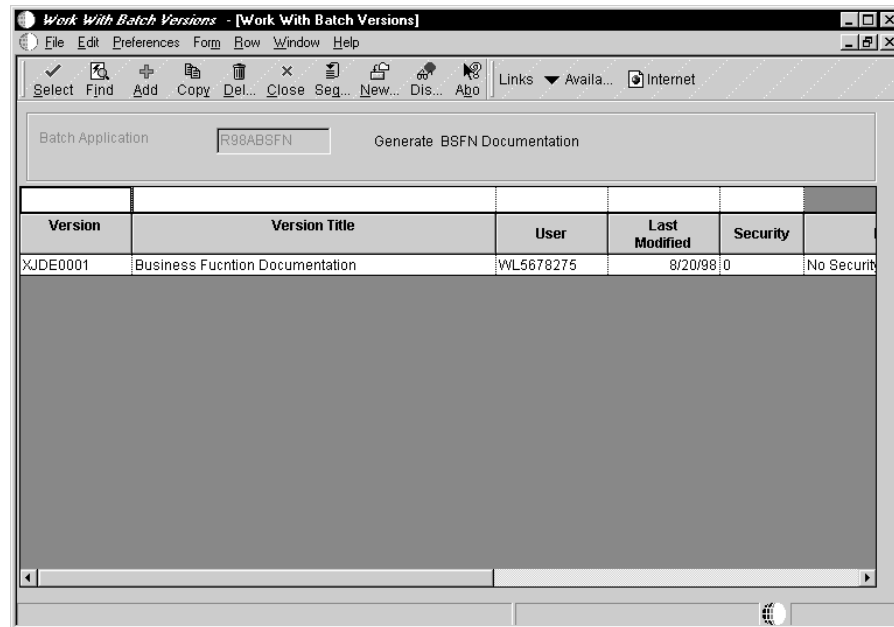
Generating Business Function Documentation

Generating business function documentation provides you with an online list of business function documentation that allows you to view documentation 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 a long time. If you create new business function documentation you may need to regenerate the business function documentation just for that business function.

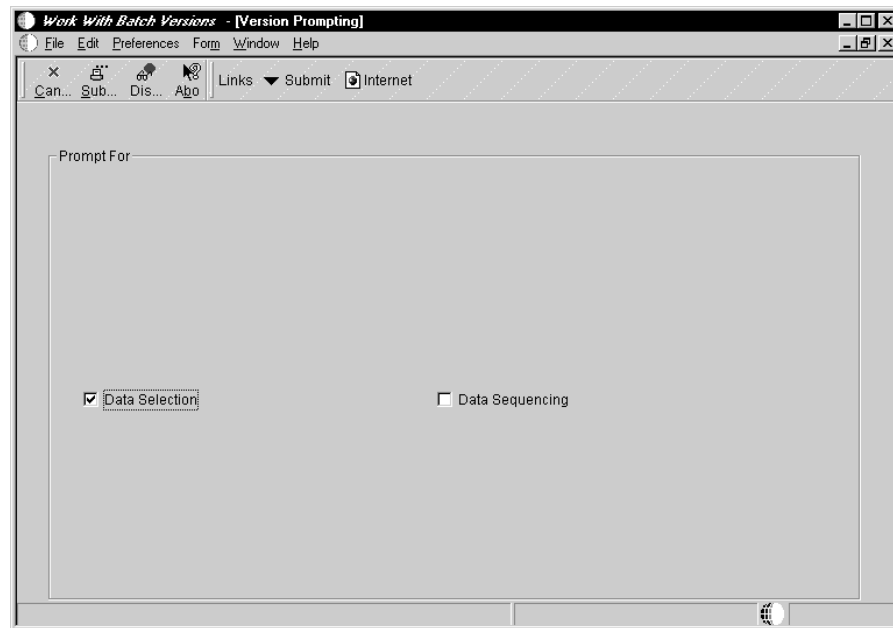
► To generate business function documentation

From the Cross Application Development Tools menu (GH902)

1. Select Generate BSFN Documentation.



2. On Work with Batch Versions, choose version XJDE0001.



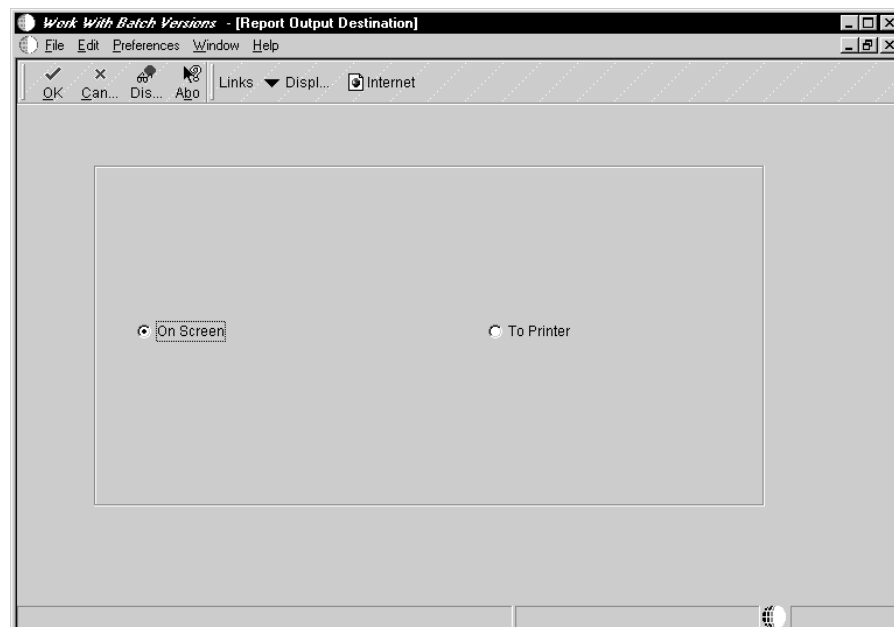
3. If you do not want to generate all business function documentation, on Version Prompting, choose the following option:
 - Data Selection
4. Click Submit.

	Operator	Left operand	Comparison	Right operand
	Where	BC Category - Business Function (F9862	is equal to	BC Business Function Category 3 (F9862

5. On Data Selection, build your criteria for data selection and click OK.
 Select only those functions for which you are generating documentation.



6. Depending on the criteria you choose, you might also need to designate processing options.



7. If you are running your report locally, on Report Output Destination, choose one of the following output destinations:
 - On Screen
 - To Printer

A hypertext markup language (HTML) link is created for each business function for which you generated documentation. An Index HTML file is also created. These

HTML files are placed in your output queue directory. Output is in the following format:

Function Name				
<i>Function Description from O/L</i>				
Parent DLL:				
Location:				
Language:				
Purpose				
Special Handling				
Data Structure				
Parameter Name	dataitem	data type	req/opt	i/o/both

Viewing Business Function Documentation

You can view your business function notes from several different locations, including:

- Business Function Search
- Business Function - Values to Pass
- Business Function Documentation Viewer

Viewing Documentation from Business Function Search

When you make a connection to a business function in event rules, the Business Function Search form appears. You can then select the function you want to call. From the row menu, choose Data Structure Notes or Attachments to view the documentation for the business function.

Business Function Search

File Edit Preferences Row Help

Select Find Close Seq... New... Dis... Abo... Links Data ... Internet

Source Language C C

	Function Description	Source Module	System	Category	Us
	A Function Two B55SM1-Test Change	B55SM1	55		
	A Test Merge Function One	B55SM3	55		
	AVR Receipt Post	B03B0067	03B	BAT	
	AAI Range Of Accounts Clear	B0000115	00		
	AAI Range Of Accounts Compare	B0000115	00		
	AB - Account Balance	B83000AB	83	BAT	CAL
	AlCU, Get Display Decimal Value from DD	B0000102	00		
	ARStatementPrintDriver	B03B0127	03B	BAT	
	Account AR Document Coche	B7600340	76		

Row:14

Viewing Documentation from Business Function - Values to Pass

You can click one of the following buttons on Business Function - Values to Pass to view documentation for a single business function (refer to *Business Function Event Rules* for more information about accessing this form).

- Business Function Notes
- Structure Notes
- Parameter Notes

Business Function - Values to Pass

Function Name:
Account Balance Calculate

Run Process
☐ Asynchronously

Available Objects

- Special Values
 - <Literal>
 - <Blank>
- Business View Column
 - BC Agreement Number - Distribution (F4211)
 - BC Unit of Measure - Entered for Unit Price (F4211)
 - BC Work Station ID (F4211)
 - BC Program ID (F4211)
 - BC User ID (F4211)
 - BC Transaction Originator (F4211)

Data Structure

Value	Dir	Data Item
BC User ID (F4211)	⊗	szAccountid
BC Company - Key (F4211)	⊗	szCompany
	⊗	szLedgerType
	⊗	jdThruDate
	⊗	szSubledger
	⊗	cSubledgerType
	⊗	szCurrency
	⊗	cIncludeUnposted

Business Function Notes Structure Notes Parameter Notes

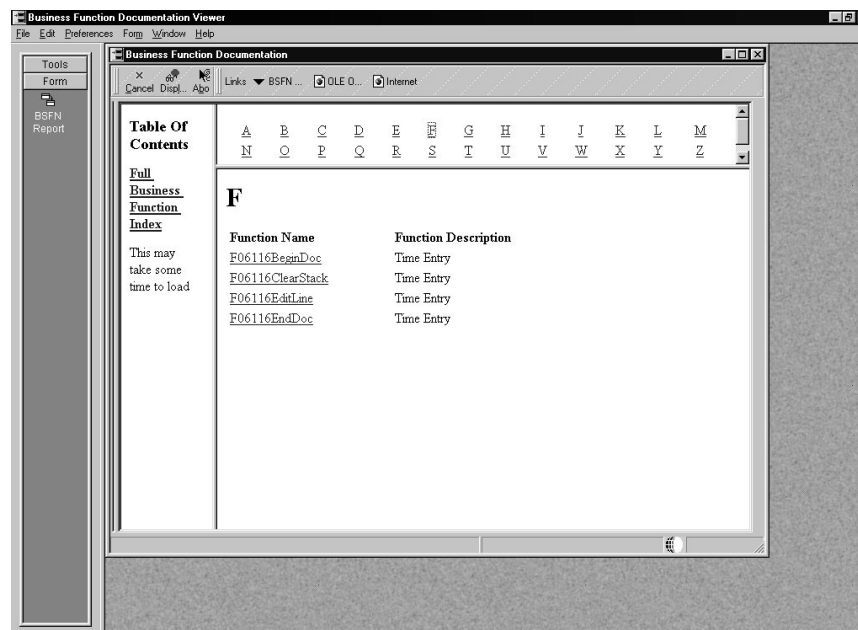
None / None

OK Cancel

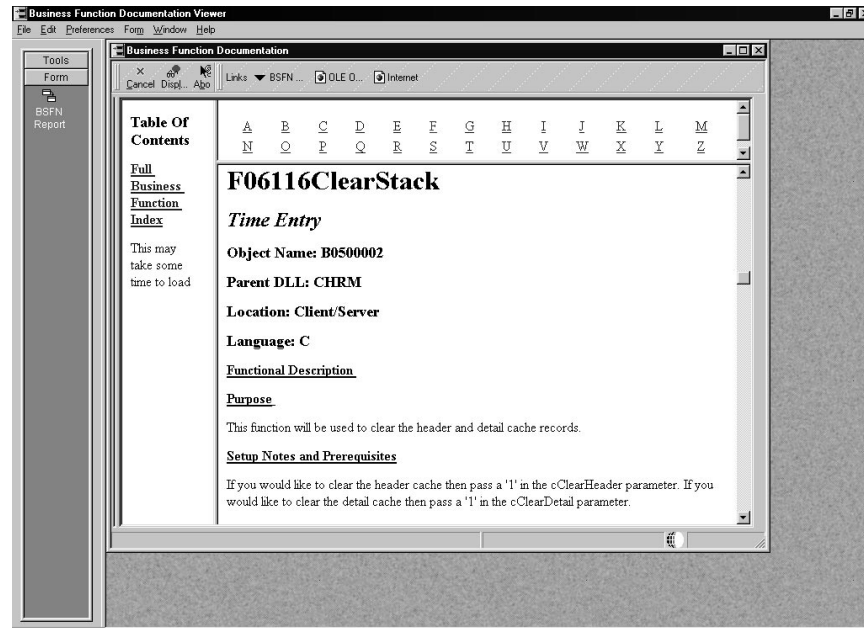
BSFN Notes	Displays the notes for the business function.
Structure Notes	Displays the notes for the whole data structure.
Parameter Notes	Displays the notes for a particular parameter.

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. Once you have generated your report, use menu GH902 to access the Business Function Documentation Viewer (P98ABSFN) to display your information. J.D. Edwards suggests that you use this method to view business function documentation.



The Business Function Documentation form contains the HTML index you generated. You can view either the entire index or select just the functions for a specific letter in the alphabet by clicking on that letter in the index. Double click a business function to view documentation 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 must be set up. This queue must point to the directory where the business function HTMLs are located. The system administrator usually generates the documentation and places it in a central directory. If you are running standalone, this path will usually be the output directory from the Network Queue Settings section of our jde.ini file. If this entry is not in your jde.ini file, it is the print queue directory in your OneWorld directory.

Caching



Caching

You can use caching to improve performance. There are two types of caching in One World.

- One type of caching is when OneWorld automatically caches certain tables, such as constants tables, when it reads them from the database at startup. OneWorld caches these tables to a user's workstation or to a server for faster data access.
- The other type of caching is when OneWorld applications are enabled to use cache. JDECACHE APIs allow the server or workstation memory to be used as temporary storage.

This section discusses the following:

- ☐ Working with JDECACHE
- ☐ JDECACHE cursors
- ☐ JDECACHE errors
- ☐ JDECACHE example program
- ☐ JDECACHE standards

When to Use JDECACHE

The following example describes when an application might use the JDECACHE APIs.

Work files are used when an application must store records a user enters in a grid until OK processing on the *Button Clicked* event. On OK processing all records must be updated to the database simultaneously. This is similar to transaction processing, but is not quite the same. For example, in the purchase order detail entry grid, if a user enters 30 lines of detail and then decides to cancel the transaction, all records in the work file will be 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 work file.

If you implement the above situation without using work files, irreversible updates to database tables occur when each row is exited. Using work files allows you to limit updates to tables so they only occur on OK button processing, and they are included in a transaction boundary. The work file 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 work file for updates and calculations.



Although using work files for the example above will function correctly, using cache will probably increase performance. You can use JDECACHE to store the records the user enters in one purchase order in local memory. The number of records you store depends on several things, including your cache buffer size for each record, the local memory size, and where the business function you use will run (server or workstation). Typically, you should not store more than 1000 records. For example, you would not want to cache your entire Address Book table in memory.

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 the application is running on. This means that a whole table can be read from a database and stored in memory. There is no limitation on the type of data, size of data, or number of data caches that an application can have, other than the limitations of the machine it is running on. Both fixed-length and variable-length records are supported. All you need to know to use JDECACHE on any supported platform is a simple set of API calls.

Data handled by JDECACHE is in RAM. Therefore, make sure it is really necessary to use JDECACHE. If you use JDECACHE, design your records and indices carefully. Minimize the amount of records that you store in JDECACHE because this is the same memory that OneWorld and various other applications are running in as well.

JDECACHE supports multiple cursors, multiple indexes, and partial keys processing. JDECACHE gives you flexibility in terms of positioning within the cache for data manipulation. This improves performance because searching within the cache can be reduced.

Performance Considerations

There are some guidelines for getting the best performance out of JDECACHE:

- If you use JDECACHE, cache as few records as possible.
- If you insert your data in random order, your inserts will go faster. This is because a tree structure is used to store and index data. If R/B tree is on the tree is always balanced.
- The fewer columns (segments) you use, the faster your searches, inserts and deletes will go. In a worst case scenario, each column has to be compared before determining whether to go further in the cache.
- The fewer records in your cache, the faster all operations will go.

You can control the size of the cache if you have problems running out of memory (RAM). The default size allocated for cache is 1 megabyte. To change the cache memory allocation, make the following changes in your jde.ini file:

```
[JDECACHE SETTINGS]
TotalCacheMemory=<MemorySize>
```

The memory size is the maximum size of the cache before it starts flushing to disk.

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 APIs are classified into the following groups:

- JDECACHE management APIs
- JDECACHE manipulation APIs

JDECACHE Management APIs

You can use the JDECACHE management APIs to manage cache by:

- Setting up the cache
- Clearing the cache
- Terminating the cache

The `jdeCacheGetNumRecords` and `jdeCacheGetNumCursors` APIs are used to retrieve cache statistics, and 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`
- `jdeCacheGetNumRecords`
- `jdeCacheGetNumCursors`
- `jdeCacheClear`
- `jdeCacheTerminate`
- `jdeCacheTerminateAll`

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 with.
HJDECURSOR	Identifies the position in the cache that is being worked with.

The set of JDECACHE manipulation APIs consists of the following APIs:

- `jdeCacheOpenCursor`
- `jdeCacheResetCursor`
- `jdeCacheAdd`
- `jdeCacheFetch`
- `jdeCacheFetchPosition`
- `jdeCacheUpdate`
- `jdeCacheDelete`
- `jdeCacheDeleteAll`
- `jdeCacheCloseCursor`

Working with JDECACHE

The JDB environment implicitly creates, manages, and destroys the JDECACHE environment. For JDECACHE to function, JDB_InitEnv and JDB_FreeEnv APIs must be called. Each cache that is used within the JDECACHE environment is associated with a JDB user. Thus, a JDB_InitUser API call must be made prior to calling any of the JDECACHE APIs.

Before you can use JDECACHE, a cache must be initialized. You must define an index before a cache is initialized. The index tells the cache which fields in a record to use to uniquely identify a cache record. You must create a separate cache for each group of data that is referenced by the same index.

The following topics are discussed:

- ☐ Calling JDECACHE APIs
- ☐ Setting up indices
- ☐ Initializing the cache
- ☐ Using an index to access the cache
- ☐ Using the jdeCacheInit/jdeCacheTerminate rule
- ☐ Using the same cache in multiple business functions or forms

Calling JDECACHE APIs

The following steps list the order in which the JDECACHE-related APIs must be called, except for Step 7, when the actual JDECACHE APIs may be called in any order:

To call JDECACHE APIs

1. JDB_InitEnv
2. JDB_InitUser
3. Create index
4. jdeCacheInit
5. jdeCacheAdd

6. jdeCacheOpenCursor
7. The rest of JDECACHE operations in any order:
 - jdeCacheFetch
 - jdeCacheOpenCursor
 - jdeCacherFetchPosition
 - jdeCacheUpdate
 - jdeCacheDelete
 - jdeCacheDeleteAll
 - jdeCacheResetCursor
 - jdeCacheCloseCursor
8. jdeCacheTerminate
9. JDB_FreeUser
10. JDB_FreeEnv

Setting Up Indices

To store or retrieve any data in JDECACHE, you must set up *one and only one* index consisting of at least one column. The index is limited to a maximum of 25 columns, which are called segments, in the index structure. A data type is provided to you 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, keep the number of segments to a minimum.

The following is the definition of the structure that holds index information:

```
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;
```

There are a few rules to follow when you create indices in JDECACHE:

- Always declare the index structure as an array that holds one element (for future enhancements).
- Always use `memset()` for the index structure.
- Always assign as elements the number of segments that correspond to the number of columns you have in the `CacheKey` array.

- Always use `offsetof()` to indicate the offset of a column in the structure containing the columns.

The following example illustrates a cache that holds the Address Book table, (F0101). The data in the cache is referenced by the index that consists of columns ABAT1, ABAC01, ABAC02, ABDC.

```
/*Declare the index array of one element*/
JDECMINDEXSTRUCT Index[1];
/*Initialize the structure*/
memset(&Index, 0x00, sizeof(JDECMINDEXSTRUCT));
Index.nNumSegments = 4;
Index.CacheKey[0].nOffset = offsetof(F0101, abat1);
Index.CacheKey[0].nSize = sizeof(f0101.abat1);
Index.CacheKey[0].idDataType = EVDT_STRING;
Index.CacheKey[1].nOffset = offsetof(F0101, abac01);
Index.CacheKey[1].nSize = sizeof(f0101.abac01);
Index.CacheKey[1].idDataType = EVDT_STRING;
Index.CacheKey[2].nOffset = offsetof(F0101, abac02);
Index.CacheKey[2].nSize = sizeof(f0101.abac02);
Index.CacheKey[2].idDataType = EVDT_STRING;
Index.CacheKey[3].nOffset = offsetof(F0101, abdc);
Index.CacheKey[3].nSize = sizeof(f0101.abdc);
Index.CacheKey[3].idDataType = EVDT_STRING;
```

The flag `idDataType` indicates the data type of the particular key. The following illustration uses ABAN8, which is a MATH_NUMERIC:

```
Index.nNumSegments = 1;
Index.CacheKey[0].nOffset = offsetof(F0101, aban8);
Index.CacheKey[0].nSize = sizeof(f0101.aban8);
Index.CacheKey[0].idDataType = EVDT_MATH_NUMERIC;
```

Initializing the Cache

After you set up the indices, call `jdeCacheInit` to initialize (create) the cache.

Pass a unique cache name so 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 a `HCACHE` handle to the cache that JDECACHE creates. This handle is in every subsequent JDECACHE API to identify the cache.

The keys in the index must always be the same for every `jdeCacheInit` 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 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 each and every JDECACHE operation.

JDECACHE makes an internal Index Definition Structure that is used to access the cache when it is populated. The following example illustrates what happens when an index is defined and passed to `jdeCacheInit`.

Example: Index Definition Structure

You decide that the cache will store records each consisting of the following structure:

```
int nInt1  
  
char cLetter1  
  
char cLetter2  
  
char cLetter3  
  
char szArray(5)
```

You decide that each of the records in the cache will be indexed uniquely by the values contained in:

- `nInt1`
- `cLetter1`
- `cLetter3`

You pass that information to `jdeCacheInit` and the following Index Definition Structure is made for internal use by JDECACHE. The Index Definition Structure is for `STRUCT` letters.

Index Key No. Index Key #1	Index Key Offset 0	Index Key Type INTEGER
Index Key #2	4	CHAR
Index Key #3	6	CHAR

Using an Index to Access the Cache

When you use an index to access the cache, the keys in the index 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. Most importantly, this means that 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 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 following example 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.

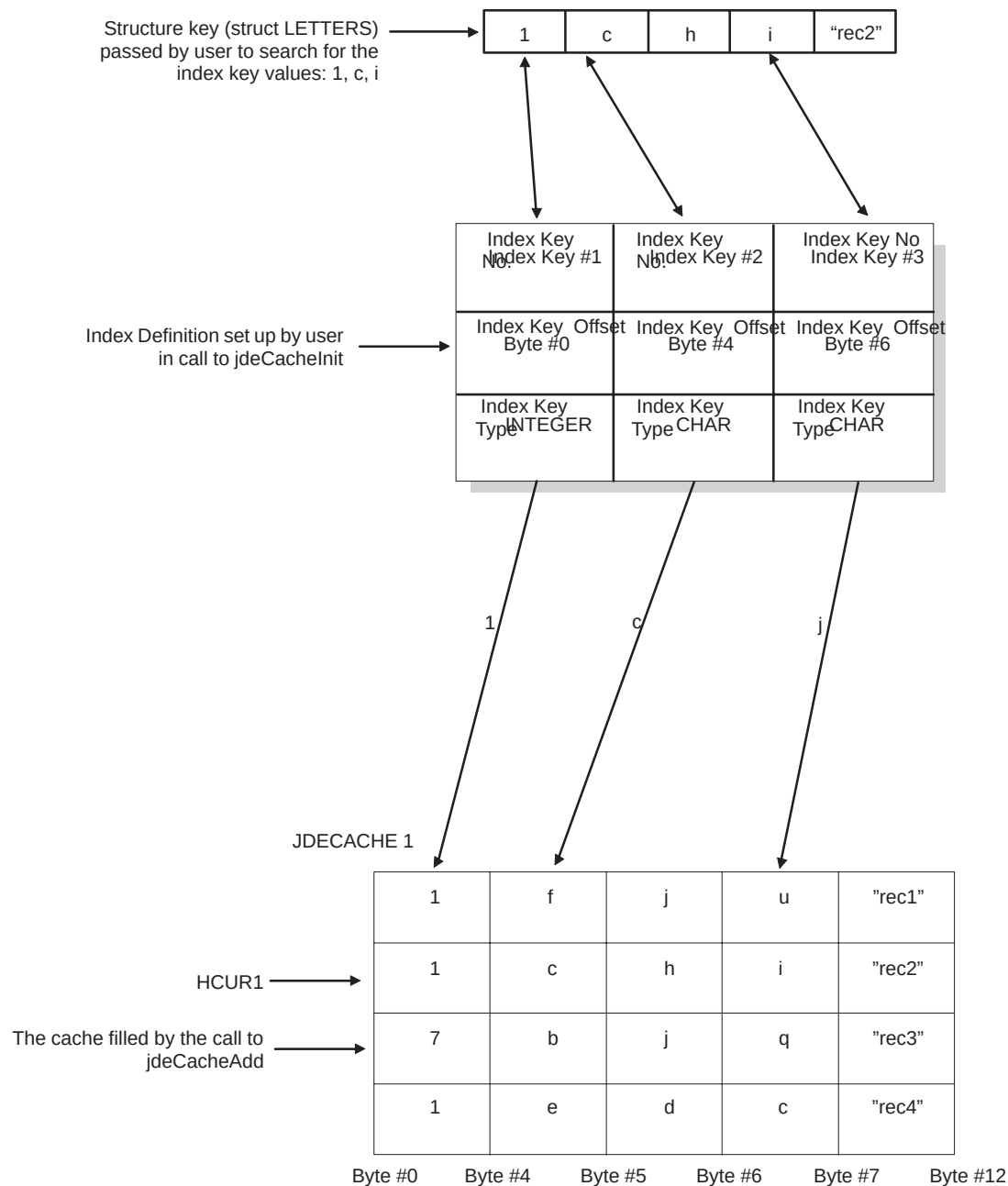
Example: JDECACHE Internal Index Definition Structure

The user is looking for a record that matches the following index key values:

- l
- c
- i

JDECACHE accesses the values you pass the structure at the byte offsets that were defined in the call to `jdeCacheInit`.

JDECACHE compares the values l, c, 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 will match the passed key will be the second structure being pointed to by `HCUR1`.



You should never create a smaller structure containing just the key in order to access the cache. JDECACHE does not store a cache record's index separately from the actual cache record like most indexing systems. 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 retrieving 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 whose offsets were defined in the index passed to `jdeCacheInit`. The structure containing the keys when accessing a cache should be the same structure used to store the cache records. The following diagram shows what will happen if any structure other than that which was used in the index definition is used to insert into the cache.

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 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.

Never access JDECACHE with compacted index keys. The key should be in the same structure type that is used to store cache records. The following example illustrates this.

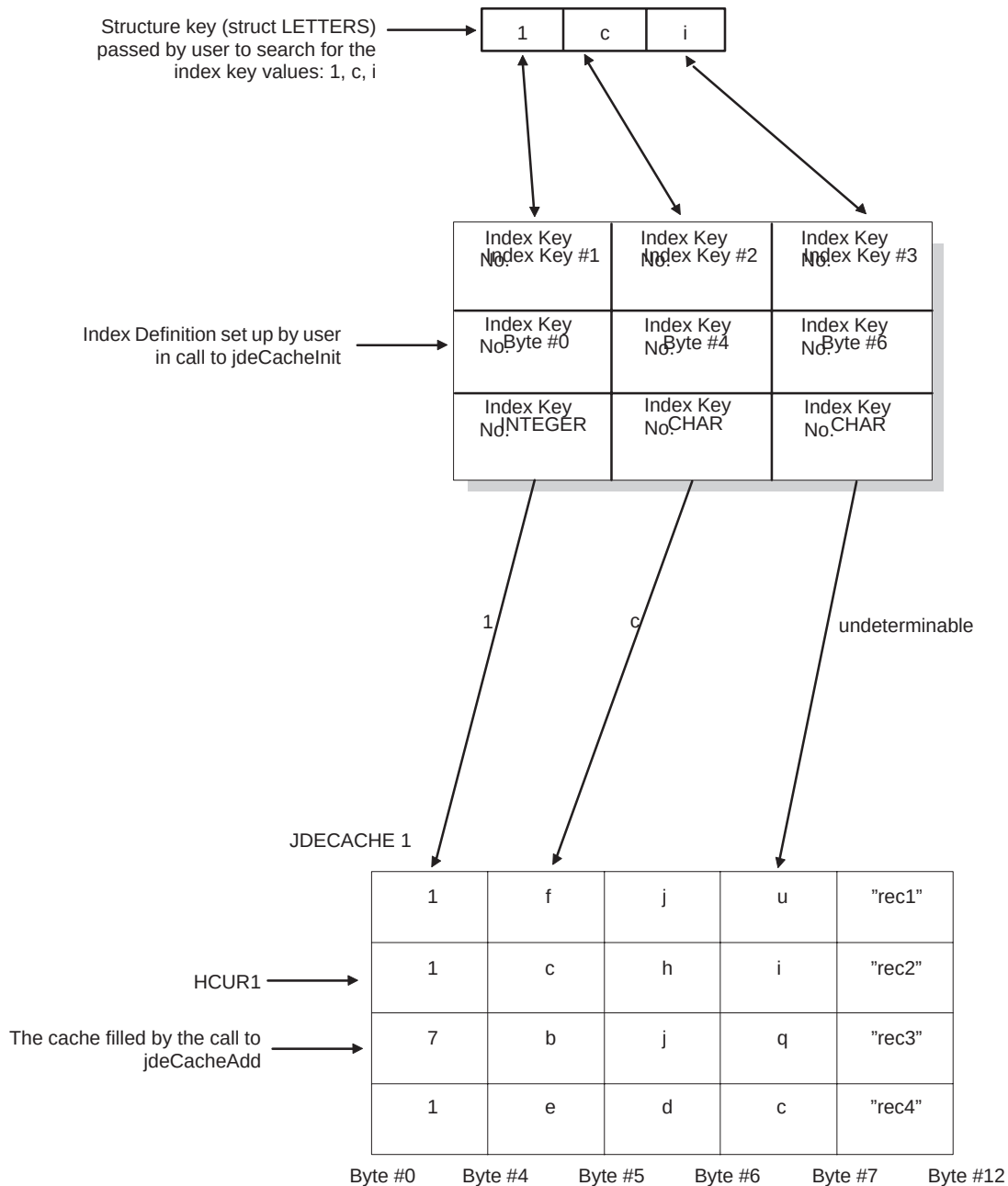
Example: Index Key Match

The user is looking for a record that matches the following index key values:

- l
- c
- i

JDECACHE accesses the values in the structure passed by the user at the byte offsets that were defined in the call to `jdeCacheInit`. The end of the structure is reached before Byte #6, so whatever is at that byte is accessed and it is undeterminable because it may contain bad data. The value `i` is not at Byte #6, but is instead at Byte #5. This structure setup is not in accordance with the Index Structure Definition.

Because Byte #6 is undeterminable, the user is not making a proper search. The user will not get the values matched as required.



Using the jdeCacheInit/jdeCacheTerminate Rule

For every jdeCacheInit there must a corresponding jdeCacheTerminate, except if the same cache is used across business functions or forms. Then all unterminated jdeCacheInit 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` passing the cache name. Before exiting that function, call `jdeCacheTerminate`. This will *not* destroy the cache. Instead, it will destroy the association between the cache and the passed `HCACHE` handle. The cache is only completely destroyed from memory 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` 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 exactly 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 exactly the same name and assigns that to you. Because the cache already has records in it, there is no 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 wish to call `jdeCacheTerminate` for the number of times that `jdeCacheInit` was called. When you initialize the same cache across business functions or forms, if you move from one form or business function to another, you will lose your `HCACHE` because it is a local variable. In order 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 that the cursor is currently pointing to is the next record that will be retrieved from the cache upon calling a cache fetch API. Thus, a cursor is a very effective way of positioning yourself in the cache for data access.

Opening a JDECACHE Cursor

JDECACHE data manipulation is cursor dependent. This means that none of the JDECACHE Data Manipulation APIs will work if no cursor has been opened. A cursor must be opened in order 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). In order to open the cursor, a call to the `jdeCacheOpenCursor` API must be made. A call to this API also makes possible the calls to all the data manipulation APIs (with the exception of `jdeCacheAdd`). Without opening the cursor, these APIs will *not* work. With this call, the cursor opens a JDECACHE Dataset within which it will work. This API does not fetch any data. It only opens the dataset. 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.

Multiple cursors to a cache can be obtained by calling `jdeCacheOpenCursor` 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, passing a `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`.

HJDECURSOR

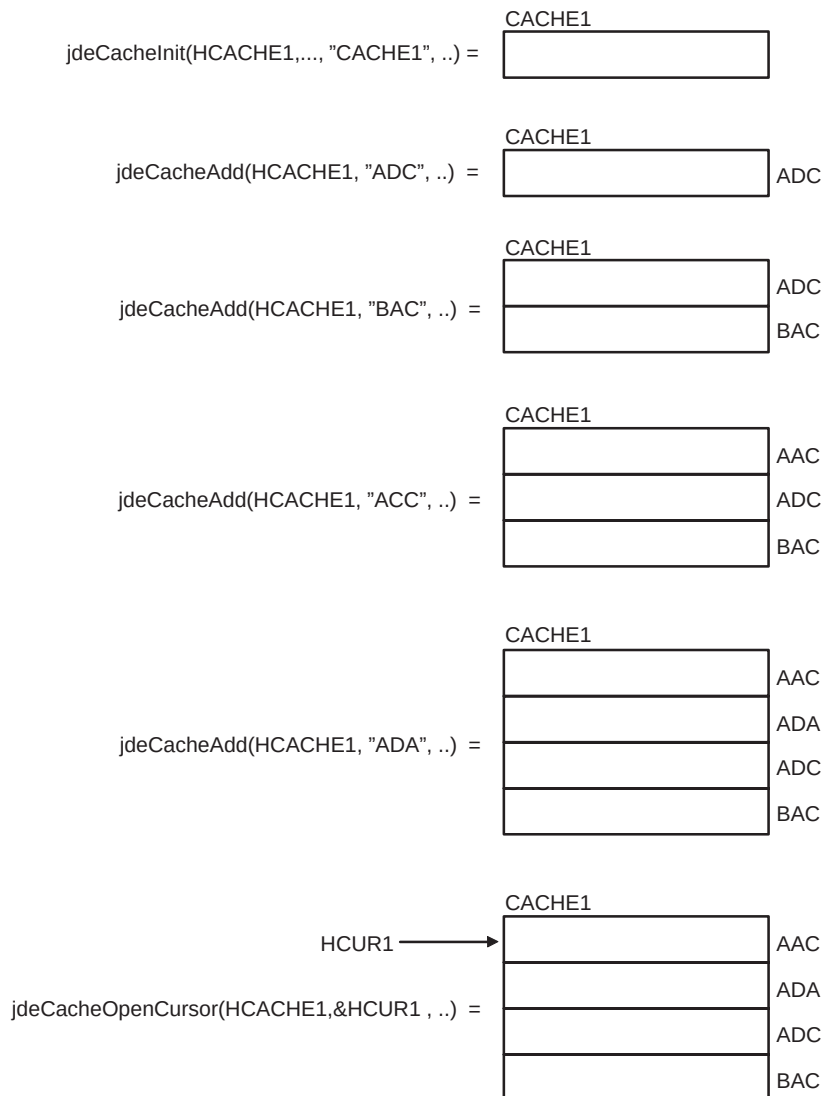
`HJDECURSOR` is the data type for the cursor handle. It must be passed to every JDECACHE Data Manipulation API except `jdeCacheAdd`.

JDECACHE Dataset

The JDECACHE dataset is defined as all 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, all records in the cache prior to the current position of the cursor are *not* considered as part of the dataset. The JDECACHE Dataset consists of the cache records sequenced in *ascending* order of the given index keys. This, in fact, means that *the order in which the records have been placed in JDEACACHE is not necessarily the order that they will be fetched* when using JDECACHE Cursors. JDEACACHE Cursors retrieves records as indicated above, in a sequential ascending order of the index keys. A forward movement by the cursor reduces the size of the dataset when performing 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.

JDECACHE Cache and Dataset Creation Example



Types of JDECACHE Cursors APIs

There are two types of JDECACHE Cursors APIs:

- Cursor-Advancing APIs
- Non-cursor-advancing APIs

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 are cursor-advancing fetch APIs:

- `jdeCacheFetch`
- `jdeCacheFetchPosition`

A call to `jdeCacheFetch` first positions the cursor to the next record in the JDECACHE dataset before retrieving it. JDECACHE Cursors also allow positioning of the cursor to a specific record within the dataset. To do this, you call the `jdeCacheFetchPosition` API, which will advance the cursor to the record matching 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 you wish to start retrieving from. 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 unless otherwise programmatically stopped.

Non-Cursor-Advancing APIs

Non-cursor-advancing JDECACHE Cursor APIs do not advance the cursor before retrieving a record, but instead keep the cursor pointing to the retrieved record. The following are non-cursor-advancing fetch APIs:

- `jdeCacheUpdate`
- `jdeCacheDelete`

Updating Records

If you want to update a specific record whose key you know, call `jdeCacheFetchPosition`, passing the known key, to position the cursor to 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`.

To update a sequential number of records, make a call to `jdeCacheFetchPosition` to get to the beginning of the sequence, if necessary. If you have made this call, then call `jdeCacheUpdate`, passing the same HJDECURSOR that you used in the call to `jdeCacheFetchPosition`. This call updates just that record that the cursor is pointing to. In order 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 your sequence.

To update a specific record whose key you know, call `jdeCacheFetchPosition` passing the known key to position the cursor to 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`.

In order to update a sequential number of records, make a call to `jdeCacheFetchPosition` to get to the beginning of the sequence, if necessary. If you have made this call, then call `jdeCacheUpdate` passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`. This call updates just that record that the cursor is pointing to. In order 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 your sequence. For each call to `jdeCacheFetch`, you need to make a corresponding call to `jdeCacheUpdate`, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetch`, to update the fetched record.

Deleting Records

Follow the exact same sequence as above, except, call `jdeCacheDelete` instead of `jdeCacheUpdate`.

The `jdeCacheFetchPosition` API

The `jdeCacheFetchPosition` API searches for a specific record in the dataset, thus, it requires a specific key. It can also be used to retrieve the first element in the cache if you pass 0 for the number of keys.

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. As mentioned above, the cursor only moves forward. To move the cursor back to the beginning of the dataset (called resetting the cursor), a call to `jdeCacheResetCursor` API must be made. It gives you a fresh JDECACHE Dataset.

Another way to reset a cursor to a specific position that is outside the current dataset (look at definition of JDECACHE Dataset) is to call the `jdeCacheFetchPosition` API.

Closing a Cursor

When the cursor is no longer needed, it must be closed by a call to `jdeCacheCloseCursor`. This closes both the dataset and the cursor. Any subsequent call to any JDECACHE API passing the closed `HJDECURSOR` without having called `jdeCacheOpenCursor` will fail.

There is no overhead associated with opening a JDECACHE Cursor for a long period of time, although you are strongly advised to close the cursor as soon as it is no longer needed.

JDECACHE Multiple Cursor Support

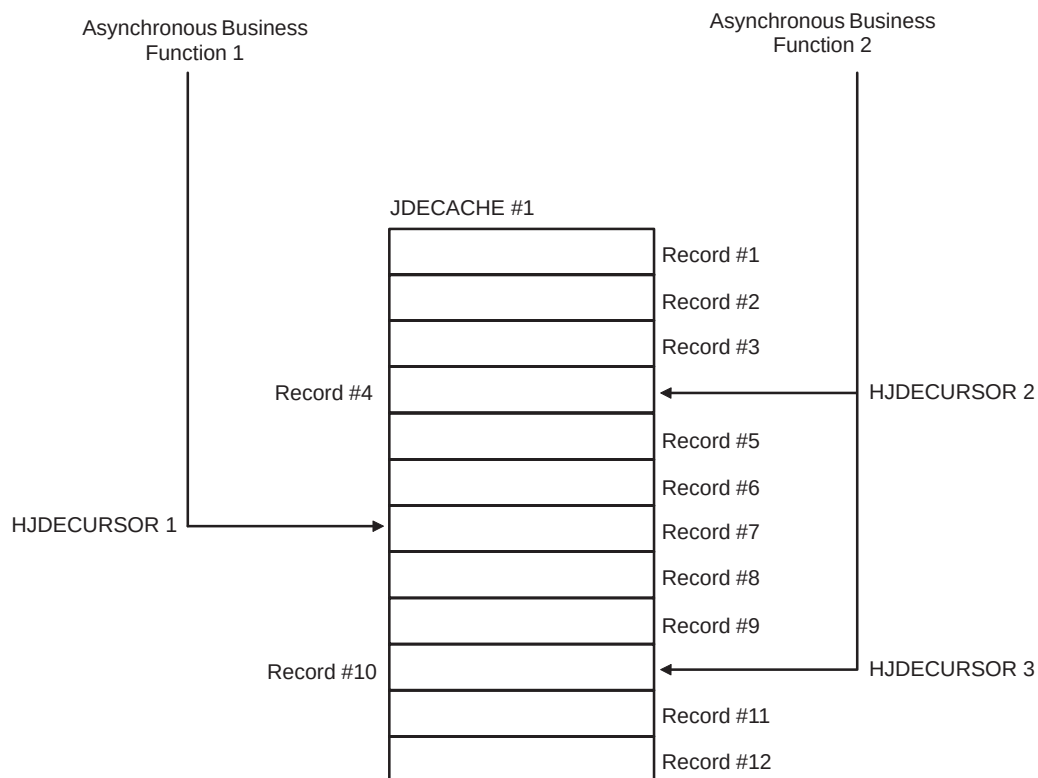
JDECACHE supports multiple open cursors. This means that each cache allows up to 100 open cursors to access it at the same time.

JDECACHE multiple cursors are designed to allow for the use of one cache by two or more asynchronously processing business functions. This means that asynchronously processing business functions can open cursors to access the cache whose relative positions within the cache are independent of each other. A cursor movement by one business function does not affect any of the other open cursors.

The use of multiple cursors has some restrictions that are mainly based on the specifications by some OneWorld applications groups.

- If you have no need for multiple cursors do not use them
- No two cursors can point to the same record at the same time unless the two cursors are both FETCHING the record.

The following diagram is an illustration of 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. This means for a defined index of N keys, the partial key is defined as the subset of the keys 1, 2, 3, 4...N-1 in that specific order. The order is very important. The partial key must be ordered in the same order as the keys are ordered in the index that was passed to `jdeCacheInit`.

For example, suppose 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 above are 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. In order to implement the JDECACHE partial key, one needs to keep in mind that the JDECACHE cursor works within a JDECACHE Dataset, which comprises the records contained within the cache *ordered by* the defined index, *the full index*. If a `jdeCacheFetchPosition` API is called passing the partial key, the JDECACHE cursor is activated and points to the first record in the JDECACHE Dataset matching the partial key. If a `jdeCacheFetchPosition` API was called, subsequent calls to `jdeCacheFetch` will fetch all the records in the dataset succeeding the fetched record *to the end of the dataset*. The cursor does *not* stop on the last record matching 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 tells JDECACHE where to start fetching from. Because the records in the JDECACHE dataset are always ordered, you are assured to get all the records satisfying the partial key first.

JDECACHE knows that you are passing a partial key because the third parameter to `jdeCacheFetchPosition` indicates the number of number of key fields that are in the key being sent to the API. JDECACHE looks at this number, and if it 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; this means JDECACHE will simply use 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 greater than the number specified on the call to `jdeCacheInit`, an error will be returned.

In order to delete using a partial key, a call to `jdeCacheDeleteAll` must be made. This will delete all the records matching the partial key. The number of key fields must also be passed to this API, in order for JDECACHE to know the partial keys being used.

Always make sure that the actual number of key fields in the structure corresponds to the numeric value describing the number of keys that must be sent to either `jdeCacheFetchPosition` or `jdeCacheDeleteAll`.

JDECACHE Errors

JDECACHE has a every effective way of handling errors. All user errors handled by JDECACHE are communicated to the user though the JDE.LOG. These errors are very comprehensive. The error statement in the log identifies the error, identifies the cause, and if the cause is not very clear, a solution is given. The following are some of the errors you might find in the JDE.LOG:

The following are some of the errors you might find in the JDE.LOG:

```
**** Module: JDECACHE   Tue Dec 10 10:31:32 1996
FILE: D:\B7\SYSTEM\jdekrrnl\misc\jdecache.c
LINE: 1821
90/138      Attempt to open a cursor on an empty cache failed
```

```
**** Module: JDECACHE   Tue Dec 10 10:34:44 1996
FILE: D:\B7\SYSTEM\jdekrrnl\misc\jdecache.c
LINE: 2495
127/130     Invalid cursor : Fetch position failed
```

```
**** Module: JDECACHE   Tue Dec 10 10:34:44 1996
FILE: D:\B7\SYSTEM\jdekrrnl\misc\jdecache.c
LINE: 2834
127/130     Invalid cursor : Cursor close failed
```

JDEDEBUG.LOG indicates each JDECACHE that was called and when it was called.

JDECACHE Error References

JDECACHE errors are used for debugging. After application development users cannot correct JDECACHE errors. JDECACHE errors are logged in jde.log. A JDECACHE error code is associated with every JDECACHE error. The error code contains information about the severity of the error, the API type it occurred in, and the error number. Following is the structure of a JDECACHE error code:

```
JDECACHE_<Severity (1 char)><API Type (2 chars)>_<Error number in
API type (0000-1000)>
```

The following values can be expected in a JDECACHE error code:

<Severity>	F for fatal errors, these are nonrecoverable errors
	S for soft errors, these are recoverable errors.
<API Type>	MG for JDECACHE management APIs
	MP for JDECACHE manipulation APIs
	IN for JDECACHE internal APIs
	AC for OneWorld memory allocation APIs
<Error Number>	A four digit number of range 0000-1000 corresponding to the API type and its severity class. This means one number can exist for the same API type but a different severity, or one number can exist for the same severity but different API type.

For example:

```
JDECACHE_FMG_0001      A fatal error has occurred in a
JDECACHE Management API and to refer to the specific error in
this API , refer to the error number 001
```

In the jde.log, the errors will appear as indicated below:

```
**** Module: JDECACHE   Sun Feb 23 19:12:46 1997  FILE:
D:\B7\SYSTEM\
jdekrnl\misc\jdecache.c LINE: 2117
136/162  JDECACHE_SMP_0001 - (jdeCacheFetch) Invalid Cache
handle
(Hint: cache terminated or cache not initialized)
```

Most of the errors associated with the JDECACHE management API type are fatal.

JDECACHE Example Program

The following example program reads the Address Book table, F0101, into a cache. The cache is indexed using the Address Book Number, which is ABAN8. With this cache each of the JDECACHE APIs is called to operate on the cached data. You can cut and paste the program at will; however, it does not print any results. You can place your own input and output calls where appropriate. This program is not meant to be meticulous artwork of C programming but rather an artful illustration of the JDECACHE APIs.

```

/*****
/*
/*      Source      :      TESTCACHE.C      */
/*
/*      Programmer   :      Chikoore, T      */
/*
/*      Date        :      12/09/96          */
/*
/*      Description  :      To illustrate the use of the jdeCache APIs
/*                          of the jdeCache APIs
/*                          which will be used to simulate the 400's
/*                          work files
/*
/*
*****/
/

#include <windows.h>
#include <windowsx.h>
#include <jde.h>
#include <F0101.H>

JDE_MEMORY_POOL GPoolCommon;
int
WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    /*Environment variables*/
    HENV          hEnv          =      NULL;

```

```

HUSER          hUser          =      NULL;

/*JDB - Related variables*/
JDEDB_RESULT   JDBcode        =      JDEDB_FAILED;
HREQUEST       hAccessRequest =      NULL;

/*JDECACHE - Related variables*/
char           cNotUsed       =      '\0';
JDECM_RESULT   jdeCachecode   =      JDECM_FAILED;
HCACHE         hF0101Cache    =      NULL;
HJDECURSOR     hF0101CacheCursor1 = NULL;
HJDECURSOR     hF0101CacheCursor2 = NULL;
JDECMINDEXSTRUCT Index[1];

/*Table - Related variables*/
ID             idTable        =      ID_F0101;
F0101          dsInsertStruct,dsRetrieveStruct,f0101;

/*Just variables*/
KEY1_F0101     dsF0101CacheKey;
short int      nNumColsInIndex =      1;
short int      nCursor1FetchCounter =    0;
short int      nCursor2FetchCounter =    0;

/*****
/*
/*                               Section 1
/*
/* Initialize the JDB and USER environments as usual
/* The cache environment is implicitly initialized by the
/* JDB_InitEnv API, you do not have to worry about it
/*
*****/

/*Initialize the JDE memory pool management utility*/
GPoolCommon = jdeMemoryManagementInit();

/*Initalize the Environment*/
JDBcode = JDB_InitEnv(&hEnv);
if(JDBcode != JDEDB_PASSED)
{
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*initialize the User*/
JDBcode = JDB_InitUser(hEnv,&hUser,"P0101",JDEDB_COMMIT_AUTO);
if(JDBcode != JDEDB_PASSED)
{
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

```

```

/*****
/*
/*                               Section 2
/*
/* Set up the index key that will be used to access the cache.
/* One and only one index is allowed per cache.
/*
/* CODE: Here the address book number (aban8) is being
/*       set up as the key.
/*
/*
/*
/*****

/*Initialize the structure*/
memset(&Index,0x00,sizeof(JDECMINDEXSTRUCT));

/*Set up the cache index*/
Index->nNumSegments = 1;
Index->CacheKey[0].nOffset = offsetof(F0101, aban8);
Index->CacheKey[0].nSize = sizeof(f0101.aban8);
Index->CacheKey[0].idDataType = EVDT_MATH_NUMERIC;

/*****
/*
/*                               Section 3
/*
/* Initialize the cache. The address of the cache handle
/* for that particular cache is passed as well as the
/* cache name and index. The cache handle will be use to
/* identify this particualr cache when calling any of the
/* cache APIs
/*
/* CODE: The cache named F0101 is being initialized
/*
/*
/*
/*****

/*Initialize the user cache*/
jdeCachecode = jdeCacheInit(hUser, &hF0101Cache, "F0101",Index);
if(jdeCachecode != JDECM_PASSED)
{
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*****
/*
/*                               Section 4
/*
/* CODE: We are going to open the address book table
/*       (F0101)and place the contents of the whole table
/*       into the F0101cache which we have just initalized
/*       in Section 3.
/*
/*
/*
/*****

/*Open the address book table*/
JDBcode = JDB_OpenTable(hUser, idTable, 0, NULL, 0 ,NULL,&hAccessRe-
quest);
if(JDBcode != JDEDB_PASSED)
{
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

```

```

/*Select all rows of the address book*/
JDBcode = JDB_SelectAll(hAccessRequest);
if (JDBcode != JDEDB_PASSED)
{
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*Initialize the structure*/
memset(&dsInsertStruct,0x00,sizeof(F0101));

/*Fetch all rows of the address book*/
while (JDB_Fetch(hAccessRequest, &dsInsertStruct, FALSE) == JDEDB_PASSED)
{
    /*Add the row to the cache*/
    jdeCachecode = jdeCacheAdd(hF0101Cache, (void *) &dsInsertStruct,
    (long int)
    sizeof(F0101));
    if(jdeCachecode == JDECM_FAILED)
    {
        break;
    } /*END IF*/
    /*Re-Initialize the structure*/
    memset(&dsInsertStruct,0x00,sizeof(F0101));
} /*END WHILE*/

/*****
/*
/*                               */
/*               Section 5       */
/*                               */
/* CODE:  We are going to open the cursors. Please      */
/*        note that cursors can only be opened after   */
/*        the cache has been filled, since the cursor has */
/*        to point to the first record in the cache     */
/*                               */
/*                               */
/*****/

/*Initialize the first cursor*/
jdeCachecode = jdeCacheOpenCursor(hF0101Cache, &hF0101CacheCursor1);
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*Initialize the second cursor*/
jdeCachecode = jdeCacheOpenCursor(hF0101Cache, &hF0101CacheCursor2);
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache, hF0101CacheCursor1);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

```



```

/*****
/*
/*                               Section 6                               */
/*                               */
/* CODE:  We are going to find the record with key 1001                */
/*        in the F0101 cache. If its not there, we add it.            */
/*                               */
/*                               */
/*****

/*Initialize the key structure. It is important that this is done*/
memset(&dsF0101CacheKey,0x00,sizeof(KEY1_F0101));

/*Set up the key*/
ParseNumericString(&dsF0101CacheKey.aban8,"1001");

/*Initialize the structure to retrieve into*/
memset(&dsRetrieveStruct,0x00,sizeof(F0101));

/*Find the cache record keyed 1001 using cursor1*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache,
hF0101CacheCursor1,&dsF0101CacheKey,nNumColsInIndex,&dsRetrieveStruct,
sizeof(F0101));
if(jdeCachecode != JDECM_PASSED)
{
    /*Initialize the structure*/
    memset(&dsInsertStruct,0x00,sizeof(F0101));

    /*Fill out the structure to insert here*/
    ParseNumericString(&dsInsertStruct.aban8,"1001");
    strcpy(dsInsertStruct.abtax, "000011171");
    strcpy(dsInsertStruct.abalph, "John Doe");
    strcpy(dsInsertStruct.abdc, "DOEJOHN");
    strcpy(dsInsertStruct.abmcu, "1001");

    /*Add the record to the cache*/
    jdeCachecode = jdeCacheAdd(hF0101Cache, (void *) &dsInsertStruct,
(long int)sizeof(F0101));
    if(jdeCachecode != JDECM_PASSED)
    {
        jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
        jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
        jdeCacheTerminate(hUser, hF0101Cache);
        JDB_CloseTable(hAccessRequest);
        JDB_FreeUser(hUser);
        JDB_FreeEnv(hEnv);
        jdeMemoryManagementTerminate();
        return 0;
    }
    /*END IF*/
}
/*END IF*/

/*****
/*
/*                               Section 7                               */
/*                               */
/* CODE:  We are going to update the record with key 1002            */
/*        in the F0101 cache.                                         */
/*                               */
/*                               */
/*****

```

```

/*Set up the key*/
ParseNumericString(&dsF0101CacheKey.aban8,"1002");

/*Find the cache record keyed 1002 using cursor1 and keep the cursor
pointing to that record because we want to update it*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey, nNumColsInIndex,&dsRetrieveStruct, sizeof(F0101));
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}

/*Update the structure we have just retrieved here*/
strcpy(dsRetrieveStruct.abalph, "Put Alpha Name Here");

/*Update the record with the record just retrieved. The cursor is already
pointing to this record so we do not need a key*/
jdeCachecode =jdeCacheUpdate(hF0101Cache, hF0101CacheCursor1,
&dsRetrieveStruct,sizeof(KEY1_F0101));
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*****
/*
/*                               */
/*           Section 8           */
/*                               */
/* CODE:  We are going to delete the record with key 1001      */
/*           in the F0101 cache.                                */
/*                               */
/*                               */
/*                               */
/*                               */
/*****/

/*Set up the key*/
ParseNumericString(&dsF0101CacheKey.aban8,"1001");

/*Find the cache record keyed 1002 using cursor1 and keep the cursor
pointing to that record because we want to update it*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey,nNumColsInIndex,&dsRetrieveStruct,sizeof(F0101));
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}

```

```

/*Delete the record keyed 1001 using cursor 1.*/
jdeCachecode = jdeCacheDelete(hF0101Cache, hF0101CacheCursor1);
if(jdeCachecode != JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache, hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache, hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*****
/*
/*                               Section 9
/*
/* CODE:  We are going to look for the record with key
/*         1001 in the F0101 cache. We have just deleted it
/*         so we should not find it.
/*
/*
/*
/*
/*
/*
/*
/*****/

/*Initialize the structure to retrieve into*/
memset(&dsRetrieveStruct, 0x00, sizeof(F0101));

/*Find the cache record keyed 1001*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey, nNumColsInIndex, &dsRetrieveStruct, sizeof(F0101));
if(jdeCachecode == JDECM_PASSED)
{
    jdeCacheCloseCursor(hF0101Cache, hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache, hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*****
/*
/*                               Section 10
/*
/* CODE:  Illustration of multiple cursors at
/*         work. We will use the two declared
/*         cursors to alternately retrieve data.
/*         Cursor 2 should alway retrieve what
/*         Cursor 1 has already retrieved.
/*
/*
/*
/*
/*
/*****/

/*First we need to reset both cursors to make
sure that we are starting from the beginning of the dataset.*/

/*Reset cursor 1*/
jdeCachecode = jdeCacheResetCursor(hF0101Cache, hF0101CacheCursor1);

```

```

if(jdeCachecode != JDECM_PASSED)
{
    /*reset failed*/
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}

/*Reset cursor 2*/
jdeCachecode = jdeCacheResetCursor(hF0101Cache, hF0101CacheCursor2);
if(jdeCachecode != JDECM_PASSED)
{
    /*reset failed*/
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
    jdeCacheTerminate(hUser, hF0101Cache);
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}

/*Clear the structure we are to retrieve records into*/
memset(&dsRetrieveStruct,0x00,sizeof(F0101));

/*Initialize the counter associated with cursor1*/
nCursor1FetchCounter=0;

/*Fetch the next record using cursor 1*/
while(jdeCacheFetch(hF0101Cache,hF0101CacheCursor1,&dsRetrieveStruct,NULL)
!= JDECM_FAILED)
{
    /*Increment the counter associated with cursor1*/
    nCursor1FetchCounter++;

    /*Check if cursor 1 has reached the max number of fetches*/
    if(nCursor1FetchCounter == 2)
    {
        /*Initialize the counter associated with cursor1*/
        nCursor2FetchCounter = 0;

        /*Fetch the next record using cursor 2*/
        while(jdeCacheFetch(hF0101Cache,hF0101CacheCursor2,
        &dsRetrieveStruct,NULL) != JDECM_FAILED)
        } /*END IF*/
    } /*END WHILE*/

    /*****
    /*
    /*                               Section 11
    /*                               */
    /* CODE:  END!!!
    /*       Do the normal clean up work.
    /*
    /*
    /*
    /*
    *****/

    /*First close all open cursor before terminating the cache*/
    /*Close open cursor 1*/
    jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);

```

```
/*Close open cursor 2*/
jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
jdeCachecode = jdeCacheTerminate(hUser,hF0101Cache);
if(jdeCachecode != JDECM_PASSED)
{
    /*This is just a check for failure otherwise its the same as below*/
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
} /*END IF*/

/*Close the table*/
JDBcode = JDB_CloseTable(hAccessRequest);

/*Free the user*/
JDBcode = JDB_FreeUser(hUser);

/*Free the environment, the cache is implicitly killed here*/
JDBcode = JDB_FreeEnv(hEnv);

/*Terminate the memory management utility*/
jdeMemoryManagementTerminate();

return(0);
}
```


JDECACHE Standards

This chapter describes the standards J.D. Edwards follows for using cache. You are encouraged to follow these standards also.

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 cache business function description will follow the business function description standards.
- The first word must be the noun Cache.
- The second word must be the verb Process.
- If this is an individual cache function, the words following Process should describe the cache. If this is a common cache function, the words following Process should describe the group to which the individual cache functions belong.

Example Individual Cache Functions

- Cache Process Rating Options and Equipment
- Cache Process Rating Shipment Pieces
- Cache Process All Components Cache

Example Group Cache Functions

- Cache Process Transportation Cache
- Cache Process Product Data Management Cache
- Cache Process Configurator Cache

Cache Business Function Description

- 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 for individual cache function description.

Cache Programming Standards

General Standards

- Cache APIs should only be used within a standardized cache function. Using the standard cache function allows for code to be easily created and maintained. The use of cache is simplified by using the standard functions.
- The cache function should contain the standard cache actions. Additional actions against cache that are specific to the cache may also be included.
- Standard functions of a cache function include initializing the cache, loading cache, retrieving records from cache, deleting from cache, modifying records, and terminating the cache.
- Two types of standard cache functions exist: the individual cache function and the group cache function. Individual cache functions are defined as a single business function that acts as the file server over a single cache. Group cache functions are defined as a single business function that acts as the file server over several predefined caches.

Terminating versus Clearing Cache

- This is application design specific. The deciding factor is whether the cache needs to stay around.
- Before cache termination, if necessary free memory of cache data structures.

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 appended with the job number.
- If the cache will 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 only contain the data structure name.

Defining the Cache Data Structure

- Group Cache Function

The data structure for the cache function is also the layout and index for the individual cache. Typedef the data structure and paste it in 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.

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 to pass the information stored in cache should follow the standard data items. Preferably, the key fields would be listed first in order of the cache index.

Cache Action Code Standards

CACHE_GET

- Open cursor.
- Fetches a single record from the cache using the jdeCacheFetch API.
- Fetch can be either with full or partial key. Number of keys used is passed to the function through the business function data structure standard dataitems.
- Cache cursor is not required to perform the fetch, and the cache cursor will not be returned after the record is fetched.
- Returns success or failure.

CACHE_ADD

- Inserts a record into cache using the jdeCacheAdd API.
- Full key must be defined, and values for all key fields must be passed.
- Duplicate records are not allowed in cache.

- Cache cursor is not required to add a record to cache, and the cache cursor will not be returned after the record is added.
- Returns success or failure.

CACHE_UPDATE

- Open cursor.
- Optionally, fetches the record for update from cache using the `jdeCacheFetchPosition` API.
- Updates the fetched record using the `jdeCacheUpdate` API.
- Full key must be defined, and values for all key fields must be passed.
- Cache cursor is not required to update a record in cache, and the cache cursor will not be returned.
- Returns success or failure.

CACHE_ADD_UPDATE

- If a record exists, the record will be deleted and replaced with the new record. If a record does not exist, the record will be added.
- First, the record is fetched using the `jdeCacheFetch` API. If a record was found, the record is deleted using the `jdeCacheDelete` API. The new record is added with the values passed through the business function data structure using the `jdeCacheAdd` API.
- Full key must be defined, and all values for the key fields must be passed.
- The cache cursor is not required, and the cache cursor is not returned.
- 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.
- If the number of keys is equal to the total number of keys in the index, a single record will be deleted from cache using `jdeCacheDelete` API.
- A set of records may be deleted from cache by defining a partial key. Records are deleted using `jdeCacheDelete` API.
- All records will be deleted from cache if the number of keys is set to zero. Records are deleted using `jdeCacheDeleteAll` API.
- Cache cursor is not required to delete a record from cache, and the cache cursor will not be returned.
- Returns success or failure.

CACHE_GET_NEXT

- If cache cursor is empty, open the cursor, then fetch the first record using `jdeCacheFetchPosition`.
- If cursor contains a value, fetch next record.
- Pass full or partial key. Number of keys must remain constant.
- Compare value of defined # of keys in previous record with the next record to determine end of file.
- Matching partial keys for the Group Cache function is more difficult.
- Close the cursor when key match fails and free the data pointer.

CACHE_TERMINATE

- Terminates the cache using the `jdeCacheTerminate` API.
- The cache will only be terminated when the number of users of the cache equals zero.

CACHE_TERMINATE_ALL

- Terminates the cache using the `jdeCacheTerminateAll` API.
- Make sure to free the data pointer.
- Regardless of the number of users accessing the cache, it will be terminated.

CACHE_CLOSE_CURSOR

- Closes the cursor using the `jdeCacheCloseCusor` API.

Group Cache Business Function Header File

The following is the cache data structure for a group cache business function. This data structure is also the definition of the Preference Profile Cache. In addition, each cache must have a function prototype definition.

```

/*****
* TYPEDEF for Data Structure
*   Template Name:      Cache Process Preference Profile Cache
*   Template ID:        D4900190A
*   Generated:          Mon Jun 02 10:54:17 1997
*
* DO NOT EDIT THE FOLLOWING TYPEDEF
*   To make modifications, use the Everest Data Structure
*   Tool to Generate a revised version, and paste from
*   the clipboard.
*
*****/

```

```

#ifndef DATASTRUCTURE_D4900190A
#define DATASTRUCTURE_D4900190A

typedef struct tagDSD4900190A
{
    char                szCacheActionCode[3];
    MATH_NUMERIC        mnNumberKeys;
    char                cSuppressErrorMessage;
    char                szErrorMessageID[11];
    ID                  idCacheCursor;
    MATH_NUMERIC        mnTransportationJobNumber;
    char                szPrefernce[3];
    MATH_NUMERIC        mnAddressNumber;
    char                szDocumentSet[4];
    char                szOptionsEquipment[11];
    MATH_NUMERIC        mnSequenceNumber;
    char                szModeOfTransport[4];
    char                szCountry[4];
    char                cIncludeExclude;
    MATH_NUMERIC        mnCarrier1;
    MATH_NUMERIC        mnCarrier2;
    MATH_NUMERIC        mnCarrier3;
    char                szFreightRateSchedule[9];
    char                szFreightHandlingCode[4];
    char                cRouteSelectionType;
    char                szBillableFreightSummaryDetail[2];
    MATH_NUMERIC        mnDistance;
    char                szUnitofMeasureDistance[3];
    char                szOutputQueue[11];
    char                szCopies[4];
    char                cLineShipmentLoadLevel;
}DSD4900190A, *LPDSD4900190A;

#define IDERRszCacheActionCode_1          1L
#define IDERRmnNumberKeys_2              2L
#define IDERRcSuppressErrorMessage_3      3L
#define IDERRszErrorMessageID_4          4L
#define IDERRidCacheCursor_5             5L
#define IDERRmnTransportationJobNumber_6  6L
#define IDERRszPrefernce_7                7L
#define IDERRmnAddressNumber_9            9L
#define IDERRszDocumentSet_10             10L
#define IDERRszOptionsEquipment_11        11L
#define IDERRmnSequenceNumber_12          12L
#define IDERRszModeOfTransport_13         13L
#define IDERRszCountry_14                 14L
#define IDERRcIncludeExclude_15           15L
#define IDERRmnCarrier1_16                16L
#define IDERRmnCarrier2_17                17L
#define IDERRmnCarrier3_18                18L
#define IDERRszFreightRateSchedule_19     19L
#define IDERRszFreightHandlingCode_20     20L
#define IDERRcRouteSelectionType_22       22L
#define IDERRszBillableFreightSummaryDetail_23 23L
#define IDERRmnDistance_24                24L
#define IDERRszUnitofMeasureDistance_25   25L
#define IDERRszOutputQueue_26             26L
#define IDERRszCopies_27                  27L
#define IDERRcLineShipmentLoadLevel_28    28L
#endif

```

```

/*****
* Business Function Prototypes
*****/
JDEBFRTN (ID) JDEBFWINAPI CacheProcessPreferenceProfile (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190A lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessLoadVehiclesCache (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190LE lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessLoadCompartmentsCache (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190LF lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingDelivery (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RA lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingShipmentDetail (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RD lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingShipmentPieces (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RP lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingOptionsEquip (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RO lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingRateSchedule (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RS lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingChargeCode (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RC lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingFreightCharges (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RF lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRatingShipmentDtl2 (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190RB lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessLoadHeaderCache (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190LA lpDS);
JDEBFRTN (ID) JDEBFWINAPI LoadUnloadUpdateLoadStopsCache (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190LD lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessSelectedRoutingEntry (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190TA lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessCarrierZone (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190TB lpDS);
JDEBFRTN (ID) JDEBFWINAPI CacheProcessRoutingOptions (LPBHVRCom lpBhvrCom,
LPVOID lpVoid, LPDSD4900190TC lpDS);

```

Individual Cache Business Function Header File

This is the header file for one of the CACHE business functions. The area under structure definitions contains data structure for CACHE and its key.

```

/*****
*      Header File:  B3200400.h
*
*      Description:   Cache Process Price Cost Header File
*
*      History:
*      Date           Programmer      SAR# - Description
*      -----
*      Author  09/26/1996   Toan Tran   1301382 - Created
*
*      Copyright (c) J.D. Edwards World Source Company, 1996
*
*      This unpublished material is proprietary to J.D. Edwards World Source
*      Company. 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 J.D. Edwards World Source Company.
*****/

#ifndef __B3200400_H
#define __B3200400_H

/*****
*      Table Header Inclusions
*****/

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

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

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

typedef struct tagDSB3200400Cache
{
    MATH_NUMERIC      mnSaleOrderLineNumber;
    MATH_NUMERIC      mnComponentItem;
    char              szBranchPlant[13];
    MATH_NUMERIC      mnConfiguratorLevelNumber;
    char              szLineType[3];
    char              cPriceRollup;
    MATH_NUMERIC      mnUnitPrice;
    MATH_NUMERIC      mnUnitCost;
    char              szAssemblyRuleDescription[31];
} DSB3200400Cache, *LPDSB3200400Cache;

typedef struct tagDSB3200400CacheKey
{
    MATH_NUMERIC      mnSaleOrderLineNumber;
    MATH_NUMERIC      mnComponentItem;
    char              szBranchPlant[13];
    MATH_NUMERIC      mnConfiguratorLevelNumber;
    char              szLineType[3];
    char              cPriceRollup;
} DSB3200400CacheKey, *LPDSB3200400CacheKey;

```

```

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

/*****
*/ TYPEDEF for Data Structure
*/ Template Name:    Cache Process Price Cost
*/ Template ID:      D3200400
*/ Generated:        Tue Feb 04 16:08:00 1997
*
* DO NOT EDIT THE FOLLOWING TYPEDEF
* To make modifications, use the Everest Data Structure
* Tool to Generate a revised version, and paste from
* the clipboard.
*****/

#ifndef DATASTRUCTURE_D3200400
#define DATASTRUCTURE_D3200400

typedef struct tagDSD3200400
{
    int                nNumberOfKeys;          /* Integer 01 - OneWorld */
    char               cCacheActionCode;       /* Everest Event Point 01 */
    char               cCacheCodeError;        /* Everest Event Point 01 */
    char               cSuppressErrorMessage;  /* Everest Event Point 01 */
    char               szErrorMessageID[11];    /* Data Item */
    MATH_NUMERIC        mnJobNumber;
    MATH_NUMERIC        mnSaleOrderLineNumber;
    MATH_NUMERIC        mnComponentItem;
    char               szBranchPlant[13];
    MATH_NUMERIC        mnConfiguratorLevelNumber;
    char               szLineType[3];
    char               cPriceRollup;
    MATH_NUMERIC        mnUnitPrice;
    MATH_NUMERIC        mnUnitCost;
    char               szAssemblyRuleDescription[31];
    ID                 idCursor;
    char               cReturnCursor;
} DSD3200400, *LPDSD3200400;

#define IDERRnNumberOfKeys_10          10L
#define IDERRcCacheActionCode_11       11L
#define IDERRcCacheCodeError_12        12L
#define IDERRcSuppressErrorMessage_13   13L
#define IDERRszErrorMessageID_14        14L
#define IDERRmnJobNumber_15             15L
#define IDERRmnSaleOrderLineNumber_16   16L
#define IDERRmnComponentItem_17         17L
#define IDERRszBranchPlant_18           18L
#define IDERRmnConfiguratorLevelNumber_19 19L
#define IDERRszLineType_20              20L
#define IDERRcPriceRollup_21            21L
#define IDERRmnUnitPrice_22             22L
#define IDERRmnUnitCost_23              23L
#define IDERRszAssemblyRuleDescription_24 24L
#define IDERRidCursor_25                25L
#define IDERRcReturnCursor_26           26L

#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 CacheProcessPriceCost (LPBHVRCOM lpBhvrCom,  
LPVOID lpVoid, LPDSD3200400 lpDS);  
  
/* *****  
 * Internal Function Prototypes  
 ***** */  
void I3200400_CreateInitCache (LPDSD3200400 lpDS, HUSER hUser, HCACHE  
*lpidCacheHandle);  
  
#endif /* __B3200400_H */
```


Additional Features



Processing Options

Processing options control how an interactive or batch application processes data. You can attach unique processing options to different versions of the same application. You can use processing options to:

- Control the path a user can travel through a system
- Set up default values
- Customize an application for different companies or even different users
- Control the format of forms and reports
- Control page breaks and totaling for reports

You can define processing options for an application that automatically display at runtime.

This section describes:

- ☐ Creating a processing options data structure
- ☐ Attaching a processing options template

In addition, you might need to create a processing option version. The procedures for creating a processing option version are comparable to 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	Categorizes processing options. This text appears on the tab for a processing option.
Comment	Optional, additional 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	Every processing option must be a data item in the data dictionary. Select data items for which you want default values automatically assigned, 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.



Be careful if you change processing option text that could conflict with processing option template changes. Template changes will not take effect until another package is built, but text changes will happen immediately.

The following list outlines how to create and implement processing options:

- Create processing options by building a list of parameters called a *template*
- Attach this template to an application and create event rules for the application to make use of these values
- Create versions of the application
- Specify how the processing options will be handled at runtime

At runtime, depending on the how you set up the application, one of the following occurs:

- The processing options appear, and the user is able to choose processing options and to supply values.
- A version list appears from which the user is able to choose a version.
- The application runs with a preselected set of options.

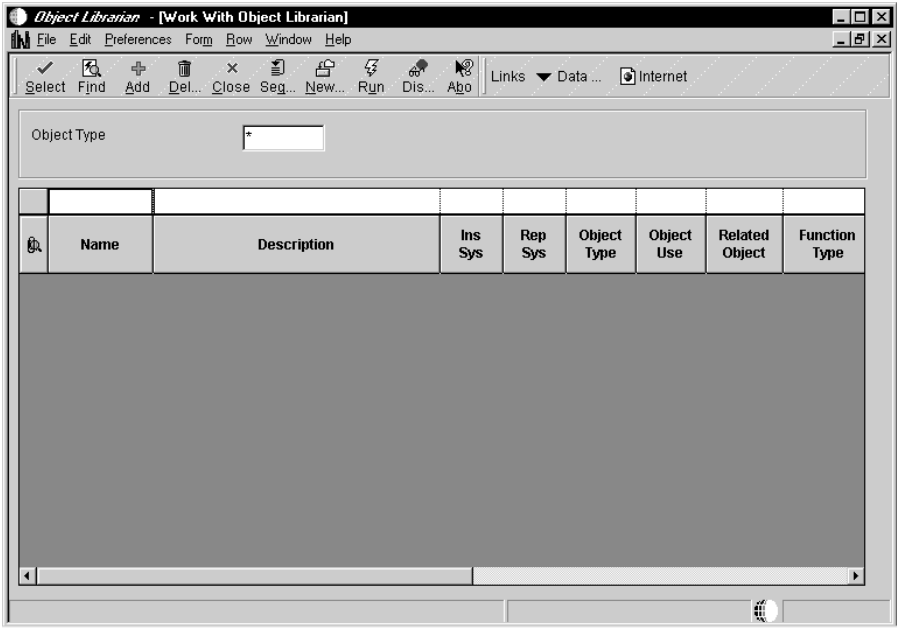
Processing option templates are stored in specification tables. Data values are stored in F983051. The processing option template and the processing option text are stored in the POTEXT TAM file until the processing option template is checked in. Then the processing option data is stored as a single T object in F98306. For batch versions, the F983051 table has an ID that points to specifications for overrides (report overrides, data sequencing, data selection, override location).

Creating a Processing Options Data Structure

You can create a processing options data structure (template) that lists the values for data items passed to the application at runtime. Any changes you make to the template reside on your workstation until you check the template in. This ensures that current users of the template are not immediately affected by your changes. Once you have checked in your changes, the next JITI will replicate the changes to the users.

► **To create a processing options data structure**

From Cross Application Development Tools (GH902), choose Processing Options Design



1. On Work with Object Librarian, complete the following field:
 - Object Type

You must enter DSTR for data structure.
2. Click Add.

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

- Object Name
- Description
- System Code
- Reporting System Code
- Object Use
- Category Code – Object Librarian 06

Make certain Structure Type is 2 to indicate a processing option template.

- SAR Number

Location	Path Code	S C	SAR	User ID	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PG	1		ER5491929		09:51:18	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
SAYLERB1	APPL_PG	3		BS5625464	8/4/98	15:29:24	

- On Parameter Design, choose Design from the Form menu.

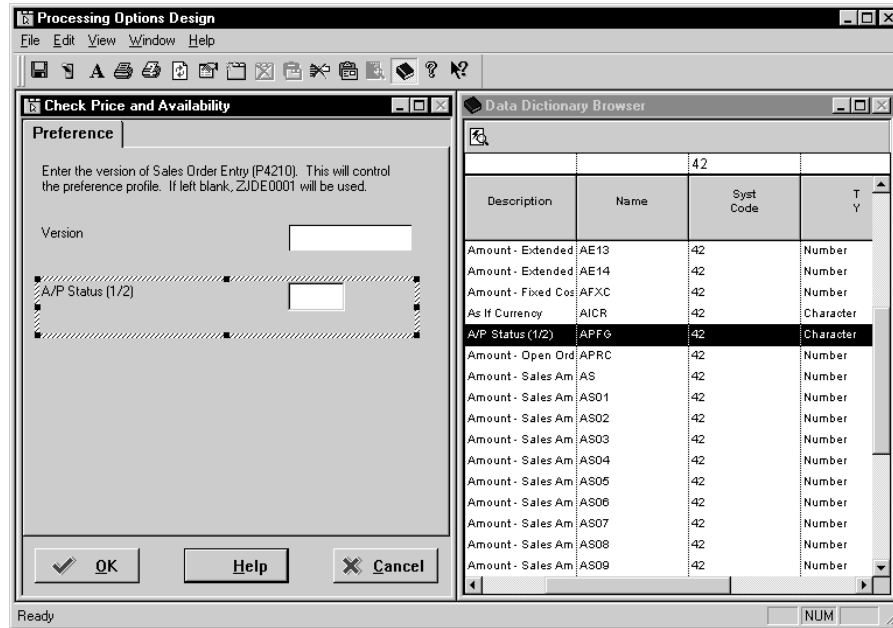
The Processing Options Design tool appears. The area on the left of the form displays how the processing option will look to the user.

Data Item	Description	Name	Syst. Code
ApprovalList	Approval List	\$APR	87
WorldVisionAutho	World Vision Autho	\$AUC	87
DateWorldVisionAu	Date - World Vision	\$AUD	87
WorldVisionAutho	World Vision Autho	\$AU1	87
DateWorldVisionAu	Date - World Vision	\$AU2	87
UserComboGuideR	UserCombo Guide	\$COR	87
WorldVisionContrac	World Vision Contrac	\$COT	87
GenerateWVAuthor	Generate WV Autho	\$GEN	87
InstallLibraryName	Install Library Name	\$ILN	87
Numberoflicenses	Number of licenses	\$LIC	87
WorldVisionLicense	World Vision License	\$LIT	87
NumberofWVPNL	Number of WV PNL	\$PNL	87
NumberPrevinstalla	Number - Prev. Inst.	\$PR#	87
NumberofWVRTS	Number of WV RTS	\$RTS	87
TotalNumberofWV	Total number of WV	\$TRT	87
UserGuideRequired	User Guide Require	\$UGR	87

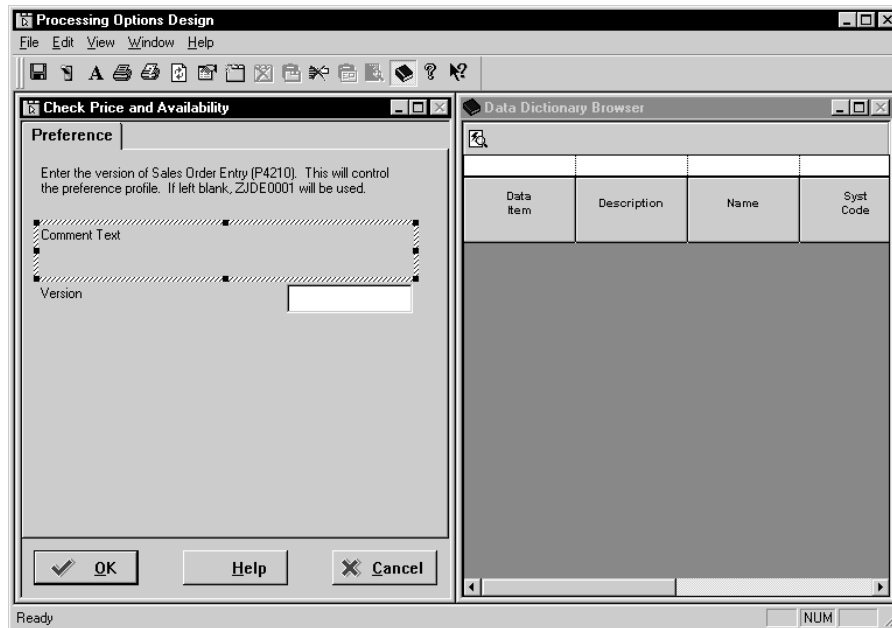
- Use one of the following methods to choose the data items you wish to add to your processing options:
 - Double-click on the item in the Data Dictionary Browser, and the item will appear in the left side of the form under the tab.

- Drag the item from the Data Dictionary Browser to the position you want it in the structure members.
6. Click an item to edit it.
- Use the hatching around the control to reposition it.
 - Select text and delete it or overwrite it.

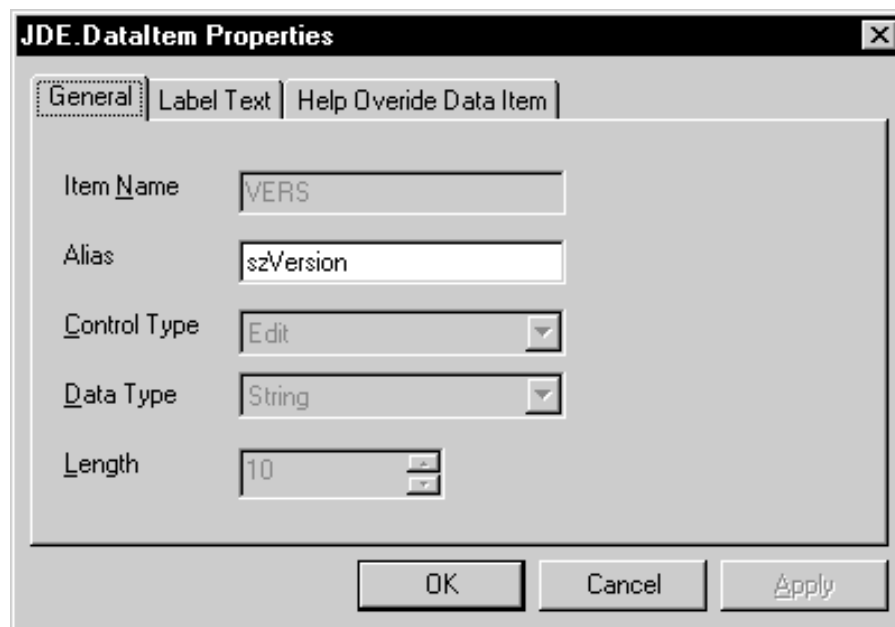
Processing Options Design automatically adjusts the size and position of data items to fit the width of the tab.



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.

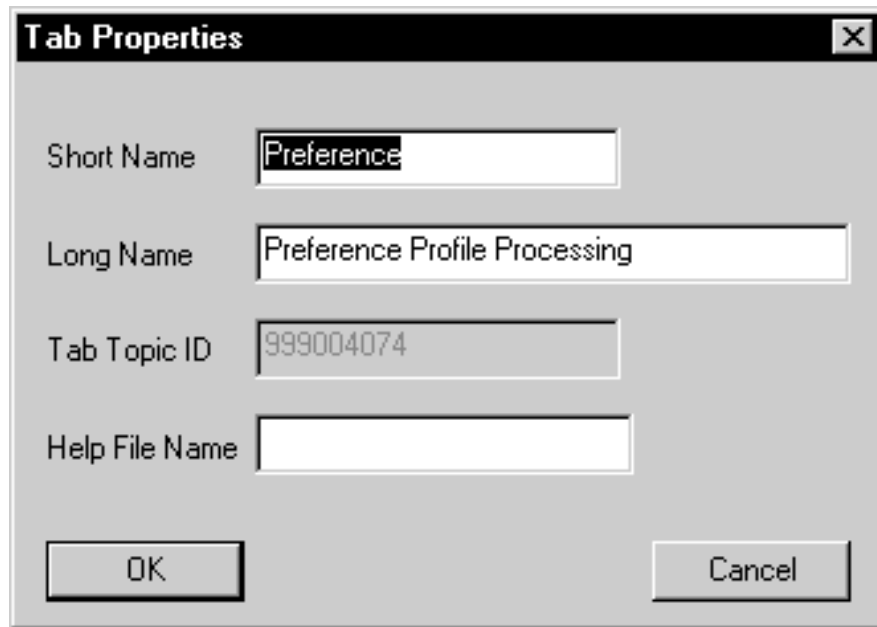


If you are on a data item, you can view its properties and change the Item Name if necessary. The Item Name should be unique.

You can click the Help Override Data Item tab to add an alternate szDict from which to get the help.

9. To view tab properties, click on the tab and choose Properties from the View menu.

You can also right-click on a tab and choose Current Tab Properties from the menu that appears.



The screenshot shows a 'Tab Properties' dialog box. It has a title bar with the text 'Tab Properties' and a close button (X). The dialog contains four text input fields: 'Short Name' with the value 'Preference', 'Long Name' with the value 'Preference Profile Processing', 'Tab Topic ID' with the value '999004074', and 'Help File Name' which is empty. At the bottom of the dialog are two buttons: 'OK' and 'Cancel'.

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.

10. 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.

J. D. Edwards Processing Option Naming Standards

A processing option includes the following four elements:

- Processing option data structure
- Tab title
- Comment
- Data item

Processing Option Data Structure

The Object Librarian name for a data structure can be a maximum of 10 or 9 characters (depending on whether you begin with T) and may be formatted as follows:

Txxxxxyyyy

T = processing option data structure

xxxxxyyyy = the program number for the application or report

For example, the 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 abbreviating tab titles wherever possible.
- Future processing options: Indicate those processing options that are currently unavailable with the word “FUTURE.” If the entire tab is unavailable, place “FUTURE” behind the extended description for the tab. If a single processing option is not available, place “FUTURE” behind the data item description.
- Each tab exists only once and is not split into multiple tabs. For example, use Process instead of Process 1, Process 2.
- The application name, such as P4310, always appears 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.”
- Application-specific tabs should be used sparingly and only when no other category makes sense. The name of an application-specific tab should be no longer than 10 characters in English to allow for translation.
- There are eight standard tab titles. Along with the extended description and processing options for each, they are as follows:

Display: Display Options, determines whether or not a field is displayed or which format of a form is displayed on entry

Defaults: Default Values, assigns a default value to a field

Edits: Data Edits, indicates whether or not a validation is to be performed

Process: Process Control, controls the process flow of the application

Application-specific tabs are:

Currency: Currency Options, contains processing options that are specific to currency

Categories: Category Codes, indicates default category codes

Print: Print Options, controls the output of a report

Versions: Versions to Execute, contains versions of the application that are called from this application

The following are standard application-specific tab titles. Group processing options under these tab titles when they apply to the type of processing specified in the tab.

Manufacturing and Distribution

Equipment Mgt - Equipment Management

Financials

Taxes: Tax Processing, contains processing options that are specific to taxes

Comment

When entering a comment for a processing option, use the following guidelines:

- Number every option on a tab. Use sequential numbering per tab, starting at 1 for each tab.

Note: When you have several processing options grouped together, you may choose to number the processing options or the comments. Choose whatever works best for the situation.

- Use action-oriented language for a processing option, such as “Enter the ...”
- Add the text “Required” to the end of the processing option if a processing option is required.
- Use a comment block where multiple processing options refer to the same topic. The comment block serves as a title for the logical group of processing options.

Data Item

When selecting a data item for a processing option, use the following guidelines:

- Where 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.
- A relevant data item should be used where the data dictionary glossary makes sense. The user can display the glossary from the processing options. Generic work fields, such as EV01, should not be used.

Field	Explanation
Application ID – OneWorld	The Application ID uniquely identifies the OneWorld Application.
Related Object	The Object Category is a functional grouping field. It is edited on the 98/OC UDC table.
Object Type	The type of object with which you are working. For example, if you are working with tables the object type is TBLE, or business functions is BSFN.
Object Use	Designates the use of the object. For example, the object may be used to create program, a master file, or a transaction journal. See UDC 98/FU.
Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Function Type	The Function Type groups business functions into Master, Major, and Minor categories based on the 98/E1 UDC table.
ER Replace	The ER Replace field indicates whether or not a business function can be replaced by Event Rules. It is edited on the 98/E2 UDC table.
ABC Priority	The ABC_Priority field indicates the status of an object as high, medium, or low. It is edited on the 98/E3 UDC table.
Process Type	The Process Type groups objects by operation, such as report, conversion, or batch process. It is edited on the 98/E4 UDC table.
Code 05	Object Librarian Category Code 05 is not in use at this time.
Ins Sys	A user defined code (98/SY) that identifies a J.D. Edwards system.
Rep Sys	A code that designates the system number for reporting and jargon purposes. See UDC 98/SY.

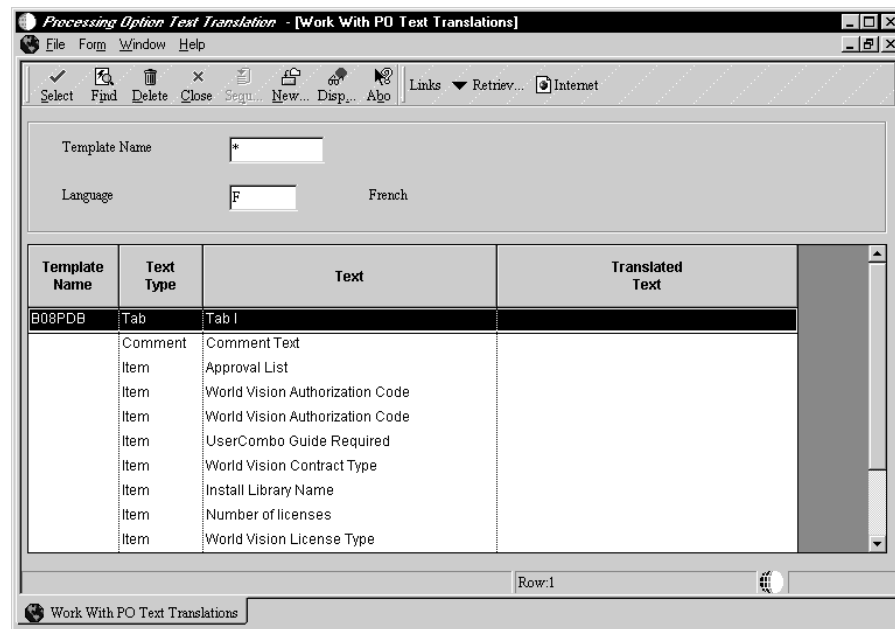
Language Considerations for Processing Options

You can change a processing options template to incorporate language features.

► To change a processing option template for text translation

From System Administration Tools (GH9011), choose Processing Options Text Translation.

1. Choose the processing option template you wish to change.



Work with PO Text Translations displays processing option text for the processing option template and language you specify in the filter fields.

2. Select the item you wish to change and enter the new text.

You can change a tab.

Processing Option Text Translation - [PO Text Translation]

File Edit Preferences Window Help

OK Cancel Disp... Apo Links Displa... Previous Next Internet

Template Name B08PDB

Domestic Language

Tab Tab 1

Long Description Tab 1

French

Tab

Long Description

Work With PO Text Translations PO Text Translation

You can change an item.

You can change a comment.

Processing Option Text Translation - [PO Text Translation]

File Edit Preferences Window Help

OK Cancel Disp... Apo Links Displa... Previous Next Internet

Template Name D1506

Domestic Language

Item 1. Enter a '1' to protect the Required Standards field.

French

Item

Work With PO Text Translations PO Text Translation

► To add a processing option template with translated text

When you add a new processing option template for an application that is language enabled, you complete the following tasks:

1. Create the application.
2. Create the Processing Template for the base language.
3. Add the language text (refer to the instructions above).

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 option template:

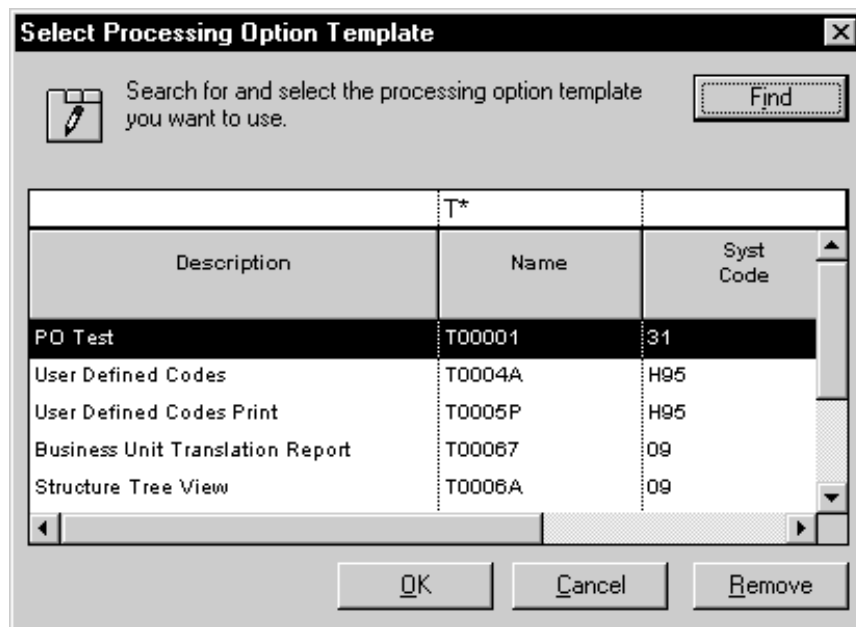
- Exists as a separate object
- Can be attached to multiple applications

When you attach a processing options template, if the processing option is designed to process on a certain event, you must attach event rule logic to enable the processing option.

► To attach a processing options template

On Application Design Aid

1. From the Form menu, choose Design.
2. From the Application menu, choose Select Processing Options.



3. On Select Processing Option Template, choose the processing option template you want to use and click OK.



The versions of an application and the event rules attached to it are dependent 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, you should first remove all existing versions of the application. You should then examine all application objects for event rules to ensure the data necessary for their operation 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 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 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 happen.
- 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 executing 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 or a rollback of a transaction occurs, the transaction is rolled back when you exit OneWorld.



Transaction processing uses commits to control database operations. Commits are commands to the database. Transactions can be auto or manual commit. For auto commits, database changes are written permanently to the database (committed) as they are executed. For manual commits, database changes are only written permanently to the database when either a commit or rollback occurs.

Commit

A commit is an explicit command to the database to permanently store the results of operations performed by a statement. This is the successful end of a transaction.

Two-Phase Commit (Manual Commit Mode)

A two-phase commit coordinate a distributed transaction. They occur 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 is the unsuccessful end of a transaction.

Any failure of an insert, update, or delete within a transaction boundary will cause 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 (for example, network problems) the DBMS performs an auto-rollback. If the user clicks Cancel on a form, a rollback command is issued through a system function.

This section describes:

- ☐ Understanding OneWorld transaction processing
- ☐ Working with transaction processing
- ☐ Understanding Record Locking

Understanding OneWorld Transaction Processing

A OneWorld 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, and a multiple-statement transaction consists of more than one statement.

You can construct a transaction within a OneWorld 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 transaction processing is on in an application, 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 OneWorld Application Design tool allows you to enable an application for transaction processing and to define what 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 on for database operations against tables that reside in DB2, then those tables must be journalled or you may have problems. Journalling can increase performance overhead due to the additional processing required. Contact your DB2 administrator if you have problems with this.

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 F0411 and F0911 tables. Because there is data interdependence between the two tables, the transaction is incomplete when there is data 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. Application starts and calls runtime engine
2. Runtime engine initializes the transaction
3. Runtime engine opens a view
4. Runtime engine performs database operations
5. Runtime engine commits database operations

If you want two connected forms to be included in the same transaction boundary you must turn on transaction processing for the parent form and designate “Include in parent” on interconnect to the second form. You do not need to turn on transaction processing for the second form because your choice on your interconnect form will override 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 changing TP On and TP Off. The table explains what would happen if you defined your transaction boundary in various ways.

Scenario		TP On	TP Off	Form, BSNF Interconnect, Table I/O	Comment
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 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).

Scenario		TP On	TP Off	Form, BSFN Interconnect, Table I/O	Comment
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 odd 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 you have a business function that calls another business function, the database operations in the function being called are still grouped within the boundaries of the parent application.

Master business functions should not define their own boundaries. You may require two or more master business functions to create one logical transaction, so the calling application should define the boundaries.

If your application calls several business functions and you need the business functions included in the transaction boundary, you must enable transaction processing. If you need to roll back database operations for the business function if there is a failure, you must designate “Include in Transaction” on the business function interconnect.



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 may cause a deadlock if you include one function in the transaction, but not the other. If you have a business function that selects records for information and also updates or inserts data to other tables, you may want to split the business function apart.

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 will be locked out until the server business function has committed. Database changes performed by server-side business functions will not be seen by the client application until the data is committed. If a server business function fails to commit or a user cancels a transaction on a server business function, the business function's transactions roll back.

Transaction Processing System Functions

There are several transaction processing system functions available. There may be situations where you need to use system function for additional transaction processing functionality.

For example, you have two forms, FormA and FormB and FormA has transaction processing enabled and calls FormB with the "Include in Parent" option on for the *Post OK Button is Clicked* event. Since FormB inherits FormA's transaction boundaries, if a user cancels on FormB, the following occurs: FormB's entries will not be written, control is returned to FormA, and FormA's entries are written and committed. If you want to prevent commitment of FormA's entries in this situation, you use 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

Refer to the Online APIs for more information about specific system functions.

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

If something occurs outside these events, it is not within the transaction boundary.

This chapter describes the tasks you must perform to work with transaction processing. It includes:

- ☐ Defining transaction processing for a form
- ☐ Extending a 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 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.

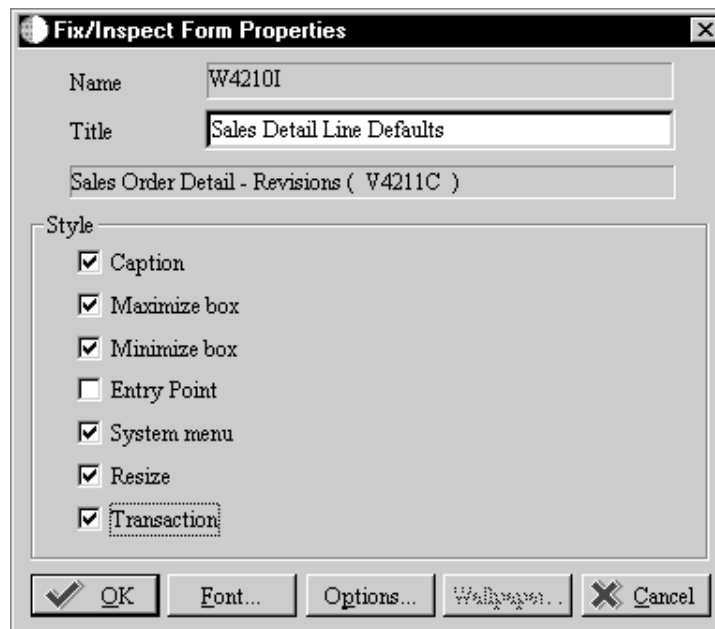
If the transaction includes data from another form, then you must extend the boundary to the applicable form through a form interconnection.

You can also extend transaction boundaries using business functions or table I/O. See *Extending a Transaction Boundary*.

► To define transaction processing for a form

On Forms Design

1. Double-click on the form to access Form Properties



2. Click the following Style option
 - Transaction

Transaction Processing for reports is done at the section level.

Field	Explanation
Transaction	Causes field data to be stored in a queue until a commit command is issued, at which time all data is moved to the table. If the transaction boundary includes other forms, check the Include in Parent option on Business Function - Values to Pass and Form Interconnect - Values to Pass.

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. You do this by enabling the Transaction Processing flag through a form interconnection in Event Rules Design.

You can do the following:

- Extend a transaction boundary between forms
- Extend a transaction boundary using business functions
- Extend a transaction boundary using table I/O

Before You Begin

- ☐ In Form Design, define form properties for each form within the transaction boundary to include transaction processing.

Extending a Transaction Boundary between Forms

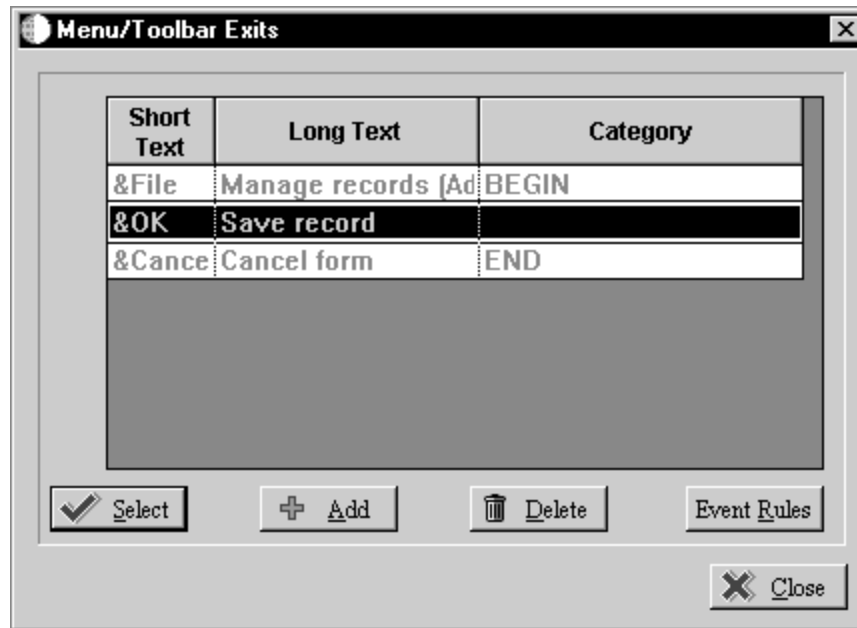
If the parent form uses manual commit, the form you connect it to will also use manual commit.



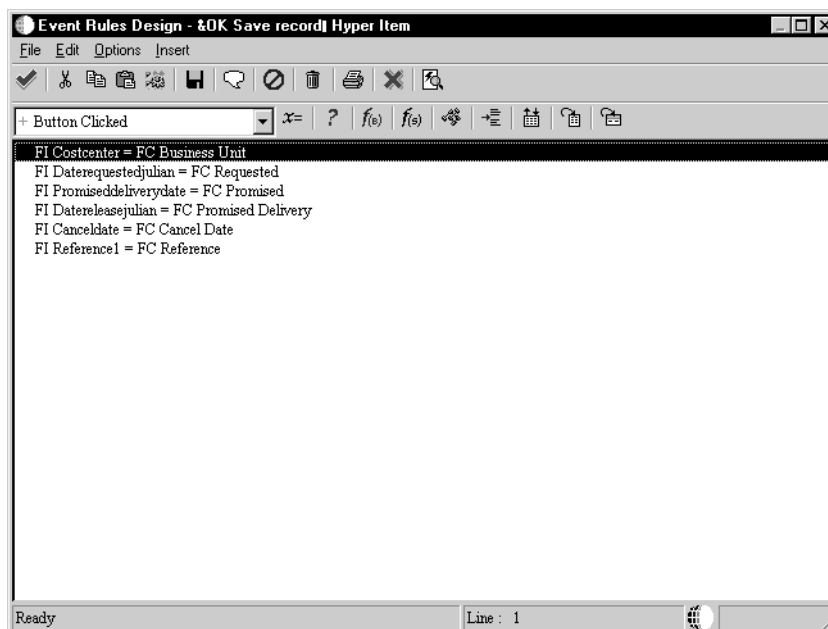
To extend the transaction boundary between forms

On Forms Design, on the parent form with which you are working

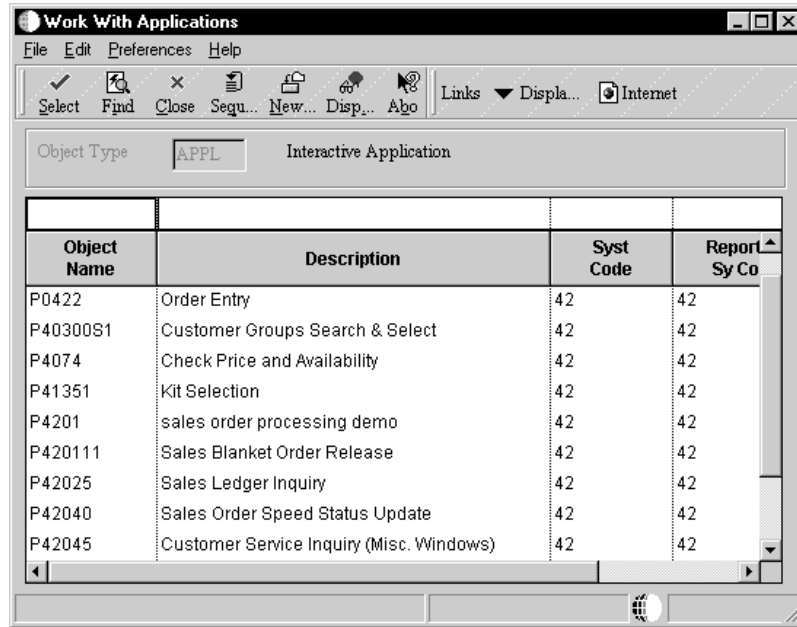
1. From the Form menu, choose Menu/Toolbar Exits.



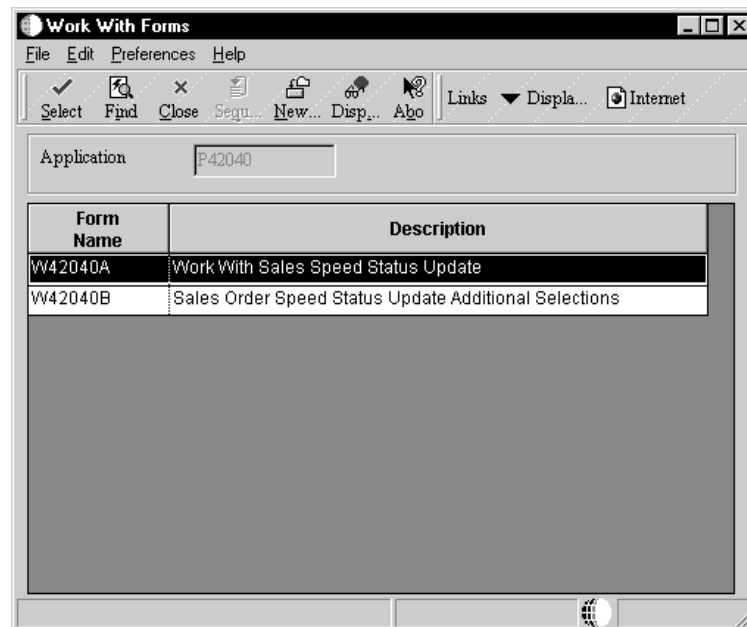
2. Choose the OK row and click the Event Rules button.



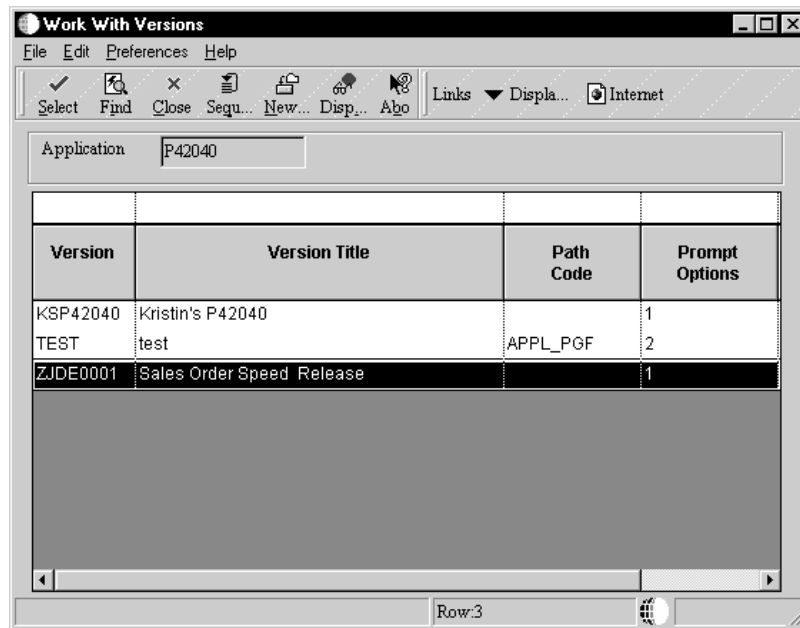
3. From Event Rules Design, choose the *Button Clicked* event, and click the Form Interconnect button.



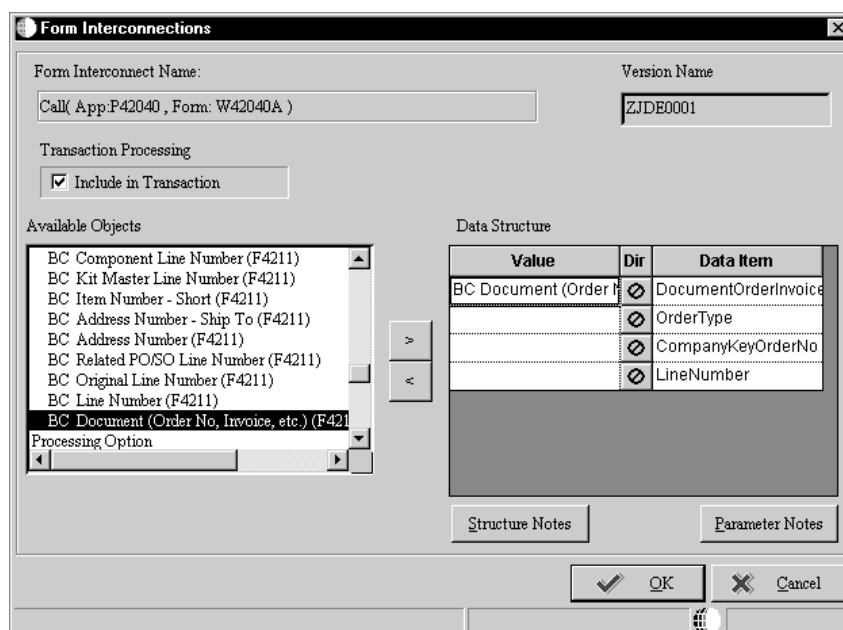
- On Work with Applications, choose the application you wish to use.



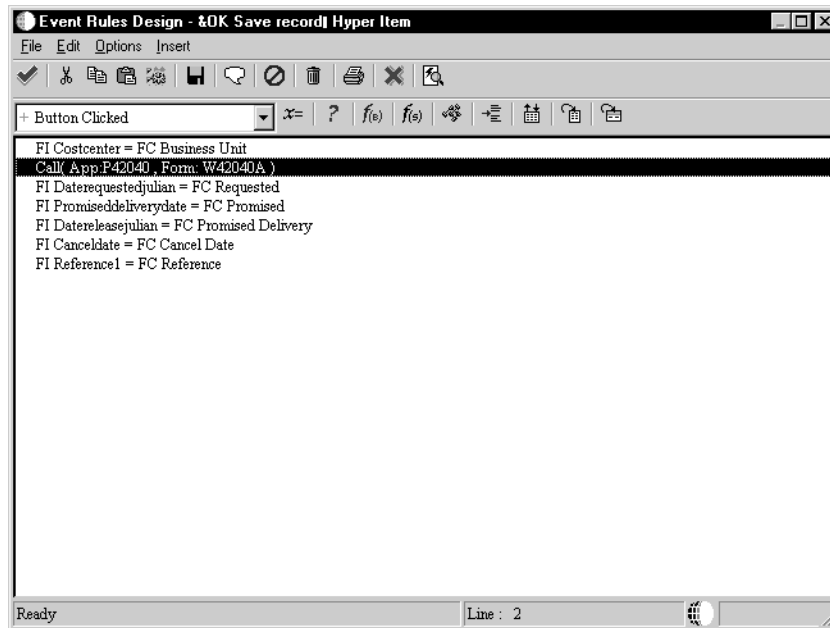
- On Work with Forms, choose the form you wish to include in the transaction boundary.



6. On Work with Versions, choose the version of the application you wish to use.



7. On Form Interconnect - Values to Pass, click the following Transaction Processing option and click OK:
 - Include in Transaction



Extending a Transaction Boundary Using Business Functions

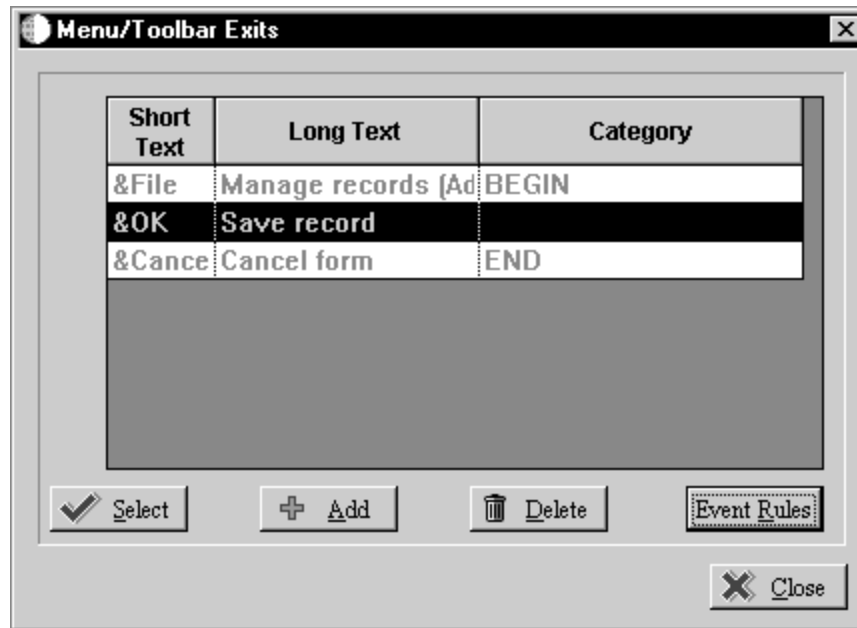
You can include a business function in a transaction boundary. If the parent form uses auto-commit, the business function you extend the transaction boundary to will also use auto-commit. Any business function not marked as “Include in Transaction” will use auto-commit. Business functions that process asynchronously can also be included in a transaction.



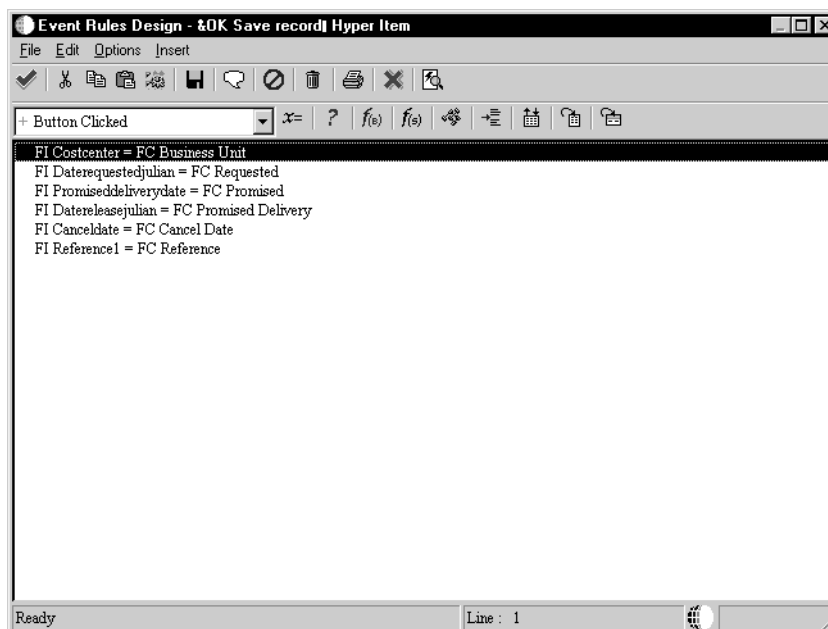
To extend a transaction boundary using business functions

On Forms Design, on the parent form with which you are working

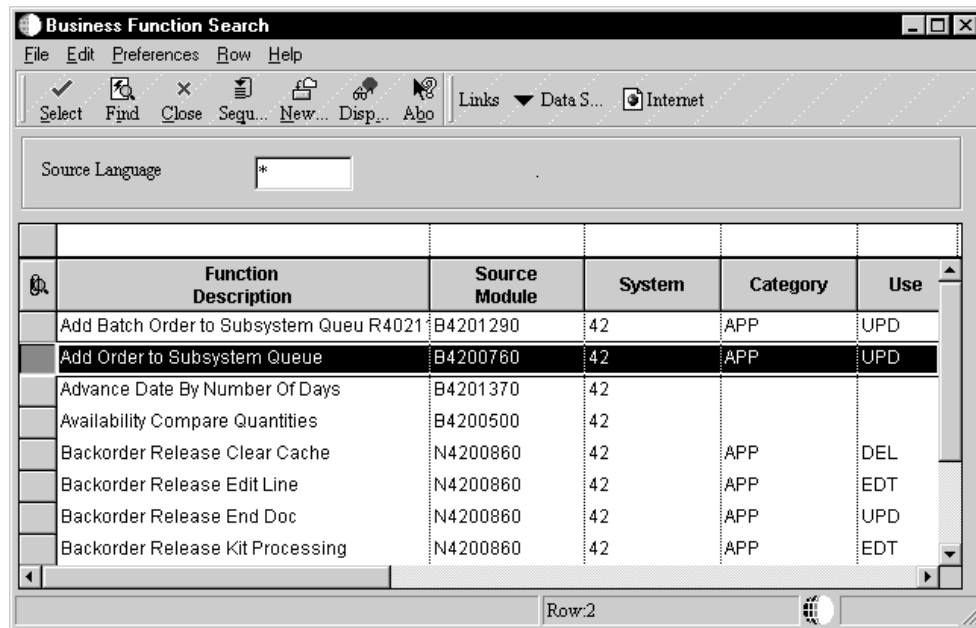
1. From the Form menu, choose Menu/Toolbar Exits.



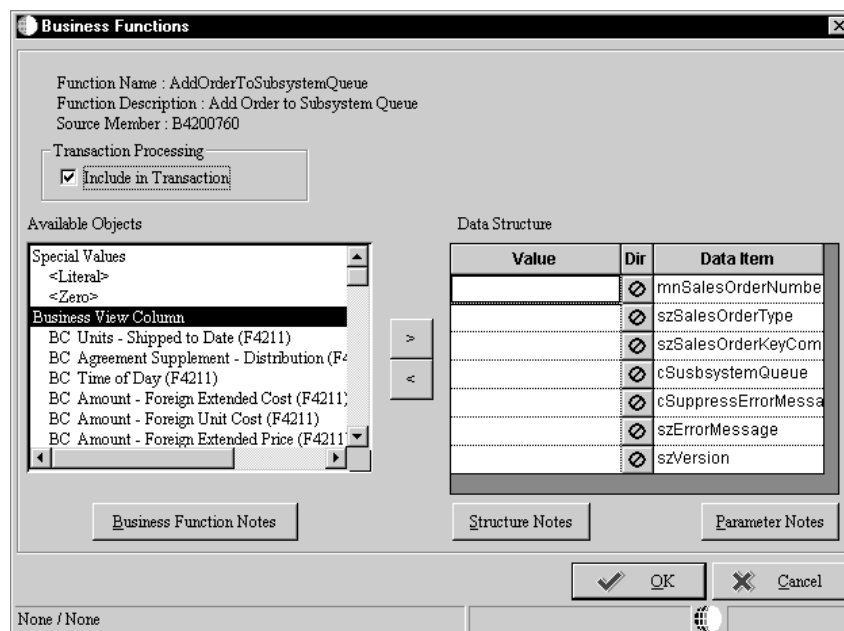
2. Choose the OK row and click the Event Rules button.



3. From Event Rules Design, choose the *Button Clicked* event, and click the Business Functions button.

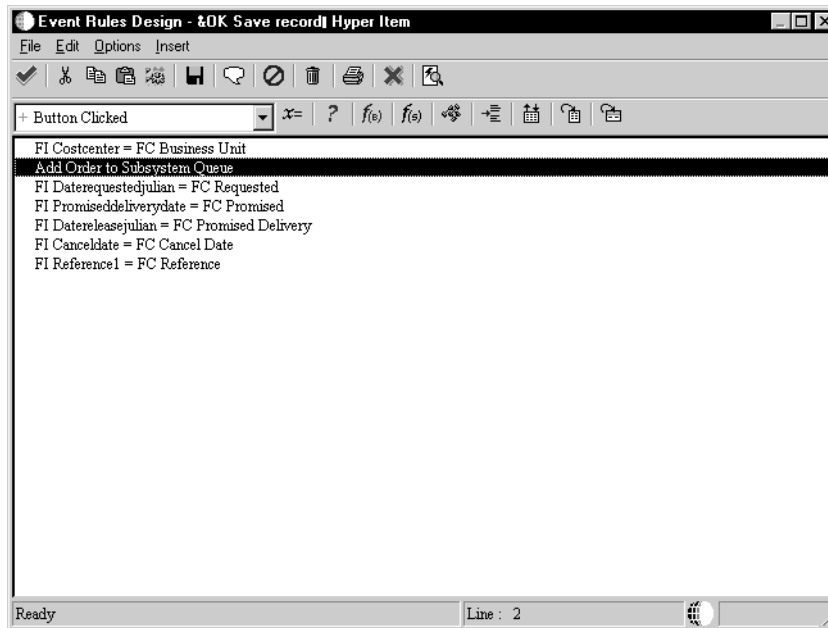


- From Business Function Search, choose the business function you wish to include in the transaction boundary.



- On Business Function - Values to Pass, 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.



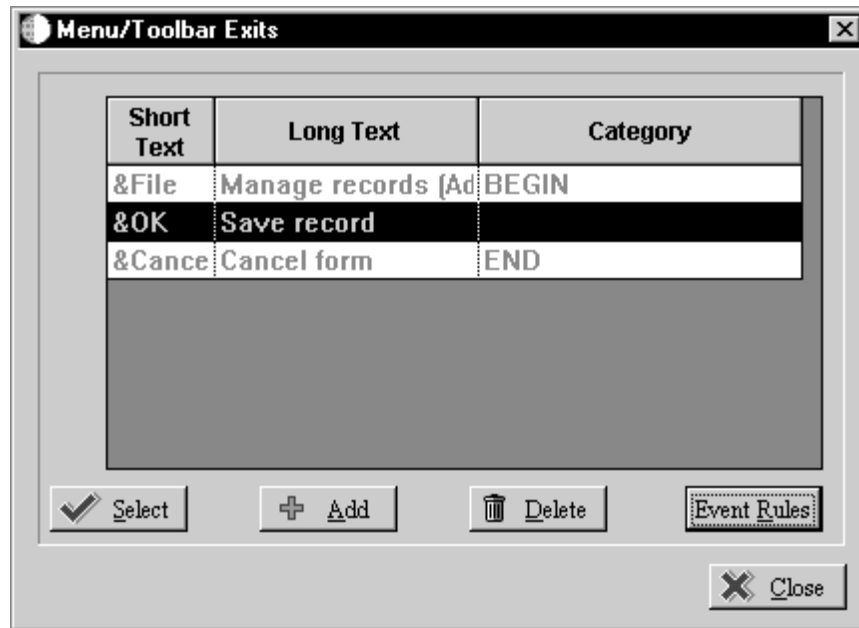
Extending a Transaction Boundary Using Table I/O

Transaction processing is only available for the Open Table operation in Table I/O. The opening of a table determines if interaction with that table will be manual commit (part of a transaction) or auto-commit. Any Open Table operation not marked as “Include in Transaction” will use auto-commit.

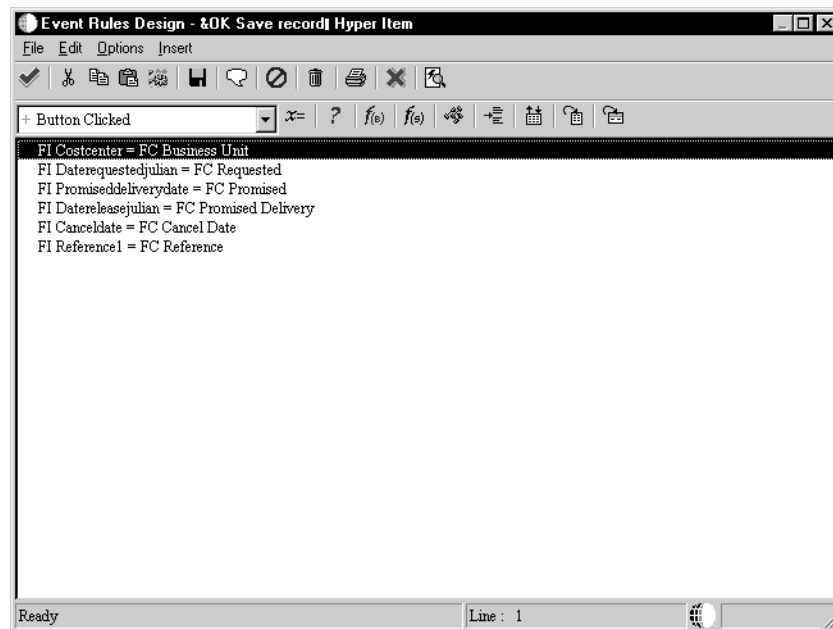
► To extend a transaction boundary using table I/O

On Forms Design, on the parent form with which you are working

1. From the Form menu, choose Menu/Toolbar Exits.



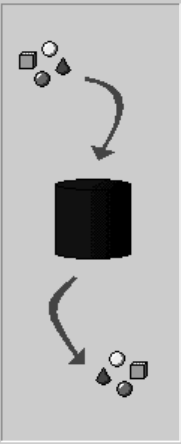
2. Choose the OK row and click the Event Rules button.



3. From Event Rules Design, choose the *Button Clicked* event, and click the Table I/O button.

Insert TableIO Operation

Select either a basic or an advanced operation, then follow the TableIO director to define parameters for the operation you selected.



Click Next to proceed...

Basic Operations:

- ☐ Fetch Single
- ☐ Insert
- ☐ Update
- ☐ Delete

Advanced Operations:

- ☒ Open
- ☐ Open Handle
- ☐ Close
- ☐ Select
- ☐ Select All
- ☐ Fetch Next

< Back
Next >
Cancel

4. On Data Source, for the Open operation, click the following option:
 - Include in Transaction

Data Source

Select a table, business view or handle.

Tables
Business Views

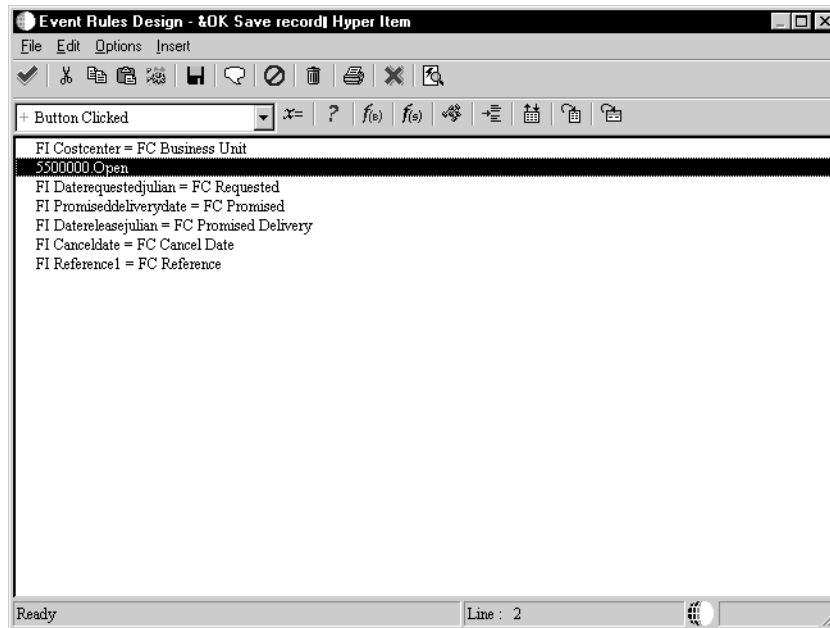
Find

Description	Name	Syst Code
Business Unit Security	F0001	00
Next Numbers - Automatic	F0002	00
Next Numbers by Company/Fiscal Year - Automatic	F00021	00
Unique Key File - Next Available Unique Key	F00022	00
User Defined Code Types	F0004	H95
User Defined Codes - Alternate Language Descriptions	F0004D	H95
Copy of F0004D for TC Testing	F0004DTC	98SA

☒ Include in Transaction

< Back
Finish
Cancel

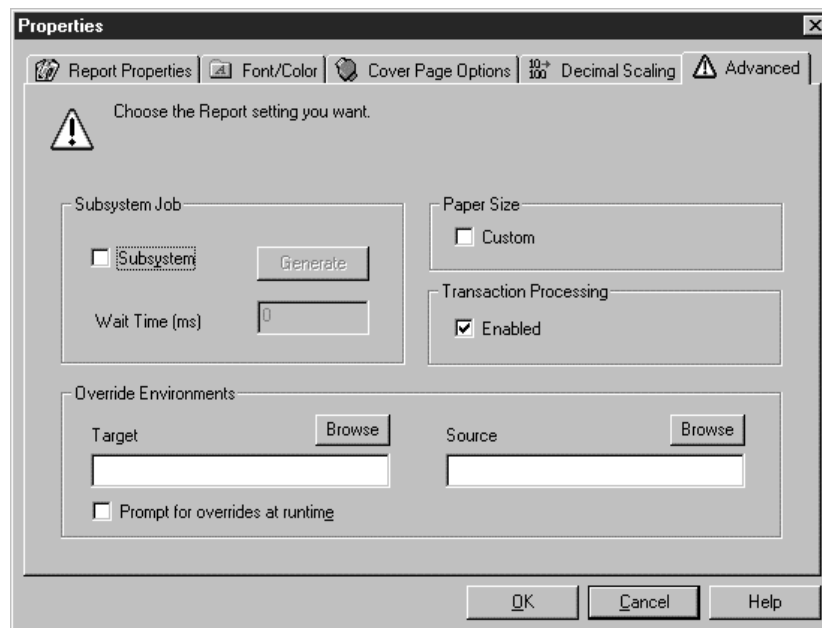
5. Click Finish.
 - The Open operation appears in your event rules.



Refer to System Functions in the Online APIs for more information about using specific system functions.

Defining Transaction Processing for a Report

In addition to interactive transaction processing functionality OneWorld also provides transaction processing for report and batch processes. To enable transaction processing for a batch process you click the Advanced tab for report properties and choose Transaction Processing.



You then use the Transaction Processing system functions to define the beginning and end of your transactions. You can also extend your transaction boundaries to include business functions and table I/O. Refer to the Online APIs for more information about these system functions.

Understanding Record Locking

OneWorld does not implement any data-locking techniques. It relies on the native locking strategy of the vendor database management system. This improves performance by reducing duplication of efforts.

There are some specific situations when the vendor database does not automatically lock as needed. In these situations, you can use explicit instructions to OneWorld to control data-locking. For example, you can use record locking to ensure the integrity of the Next Numbers facility.

OneWorld data locking may be accomplished using one of the following methods:

- Optimistic Locking

You can 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 they want to update the record.

- Pessimistic Locking

You can use record 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 “record change detection” within the workstation jde.ini file. Refer to *Setting jde.ini Files for Transaction Processing* in the *System Administration Guide* for more information. This type of database locking prevents a user from updating a record that changes during the time the user is inquiring on it. If the record has changed, the user must select the record again and then make the change. This functionality is available for business functions, table I/O and named event rules.

The following example illustrates “record change detection”. For example, suppose two users are working within the Address Book application:

Time	Action
10:00	User A selects Address Book record “1001” to inspect it.

Time	Action
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. OneWorld updates Address Book record “1001” with the update that User B made.
10:15	User A updates a field in Address Book record “1001” and clicks OK. OneWorld does not update Address Book record “1001,” and the system displays a message informing User A that the record has changed during the time User A was viewing it. For User A to change the record, User A must re-select it and perform the update.

When “record change detection occurs”, OneWorld displays the following message:

“The record you are updating has been changed by another user. Please reselect your data.”

Pessimistic Locking

Pessimistic locking is sometimes referred to as simply “record locking.” You can use “record locking” to prevent multiple users or applications from trying to update the same record at the same time. For example, suppose a user enters a transaction that uses Next Numbers. When he clicks OK, the Next Number function selects the appropriate Next Number record, validates that this number is not already in the transaction file, and then updates the Next Number record by incrementing the number. If another process tries to access the same Next Number record before the first process has successfully updated the record, the Next Number function will wait until the record is unlocked, and then complete the second process.

Record locking in OneWorld is implemented by calling published JDEBase APIs. When you use “record locking”, you should consider the time it takes 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 may or may not be part of a transaction. Record locking APIs are independent of the transaction and its boundaries. They always lock whether you are in manual or auto commit mode.

Records that are updated using record locking APIs (for example 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 record 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 Record Locking Within a Transaction Boundary

You may need to use record locking in conjunction with transaction processing. For example, if you want records locked between the read and the update you must use record locking.

Business Functions and Pessimistic Record Locking

You may want to use record 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 keep records locked for as short a time as possible. Try to ensure that the select or fetch for an update occurs as close to the update as possible.

The following pessimistic record locking APIs can be called from business functions:

- JDB_SelectAllForUpdate
- JDB_SelectForUpdate
- JDB_SelectKeyedForUpdate
- JDB_SelectKeyedPositionForUpdate
- JDB_FetchForUpdate
- JDB_FetchKeyedForUpdate
- JDB_UpdateCurrent
- JDB_DeleteCurrent

See Also

- The online *Published APIs*



Currency

Enterprises doing business internationally have additional accounting needs and added complexity. This arises from doing business in different currencies and having to follow different reporting and accounting requirements. There are some fundamental requirements for an enterprise operating internationally, including:

- Conversion of foreign currencies to their local currency
- Conversion of the different local currencies into one currency for reporting and comparisons
- Adhering to regulations defined in the countries they operate in
- Revaluation of currencies due to changes in exchange rates

OneWorld Currency Implementation

OneWorld currency implementation includes the following:

- Currency retrieval is done through database triggers and table event rules
- Currency retrieval logic is handled in Business Functions
- System APIs assist you in accessing cached tables

Advantages

OneWorld allows you, the developer, to control currency retrieval. Allowing you to control currency, instead of the system, provides greater flexibility and easier maintenance. Following are some of the advantages to allowing you to control currency:

- 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 a system module assuming 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 applications that use the table with currency business functions attached will get the same logic, so there is no need to modify each application.
- No hard-coded logic is embedded in the runtime engine.

This section describes the following:

- ☐ Working with Currency

Working with Currency

Currency implementation is needed to adjust decimal placement on Math_Numeric currency fields according to a specified 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. Common applications of currency implementation include conversion of currency amounts and revaluation of currency due to changes in exchange rates.

Implementing currency involves the following steps:

- Perform currency setup
- Create a business function that contains logic to retrieve currency information. These special currency business functions are known as currency triggers.
- Attach a currency trigger to the *Currency Conversion* event in Table Event Rules (TER).
- After designing the TER functions through Event Rules Design, the event rules will be converted to C and compiled into a consolidated DLL through the Object Configuration Manager (OCM) Application.
- Modify applications if necessary.

The JDB APIs will then be responsible for calling the appropriate TER function when the *Currency Conversion* event is triggered.

This chapter describes the following tasks:

- ☐ Setting up currency conversion
- ☐ Showing currency-sensitive controls
- ☐ Creating a currency conversion trigger

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 table F0411, the Build Triggers option will create a C source member called F0411.c. You can browse through the C code and make sure all the parameters are set up correctly. An error log will be generated if anything went wrong during the ER-to-C conversion. The error table is called eF0411.log.
- Compiles the new functions and add them into JDBTRIG.DLL. (This is the Consolidated DLL that contains TER functions).

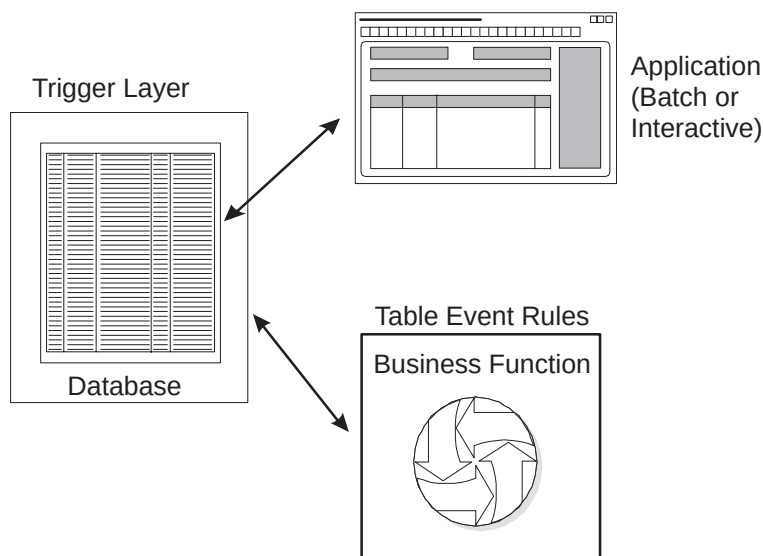
How Table Event Rules Work with Currency Processing:

The *Currency Conversion* event runs if currency processing is ON.

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.

Currency Conversion



On FETCH:

Application requests data...
 Is currency on?
 Yes, run currency trigger
 Currency Trigger calls TER
 which executes business function
 which performs business logic
 and scrubs data accordingly
 Return data to application

On ADD/UPDATE:

Application sends data...
 Is currency on?
 Yes, run currency trigger
 Currency Trigger calls TER
 which executes business function
 which performs business logic
 and scrubs data accordingly
 Update database

Application

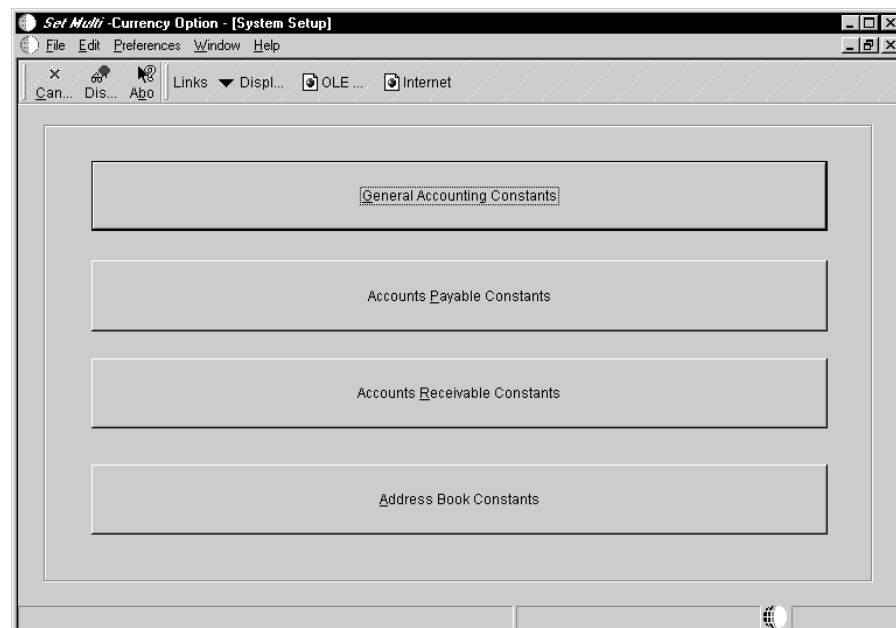
When passing Math_Numeric currency fields into a business function, the currency values in the respective data structure must be populated. Math_Numeric work fields that contain currency values also need the proper currency information.

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.

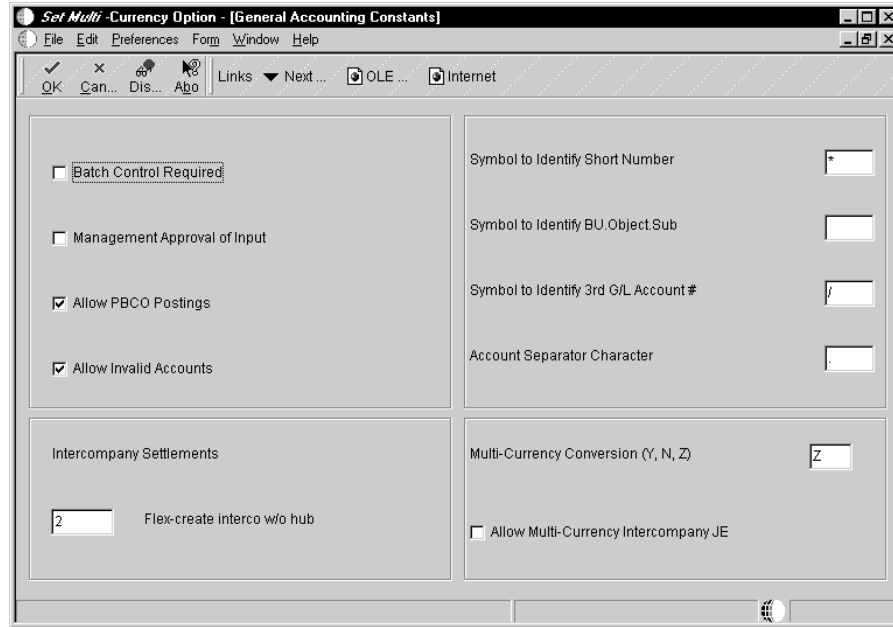
Setting Up Currency Conversion

► To set up currency conversion

1. From the Multi-Currency Setup menu (G1141) choose the Set Multi-Currency option.



2. Click General Accounting Constants.



3. On General Accounting Constants, enter a value in the Multi-Currency Conversion field.

Valid values are:

N	Do not use Multicurrency accounting
Y	Turn Multicurrency on and use multipliers to convert currency. The system will multiply the foreign amount by the exchange rate to calculate the domestic amount.
Z	Turn Multicurrency on and use divisors to convert currency. The system will divide the foreign amount by the exchange rate to calculate the domestic amount.

The currency conversion flag is stored in Company Constant table F0010 (CRYR field with Company '00000')

You can set the Multi-Currency Conversion option to “N” so that no currency conversion is done in JDB and the runtime engine.

Showing Currency-Sensitive Controls

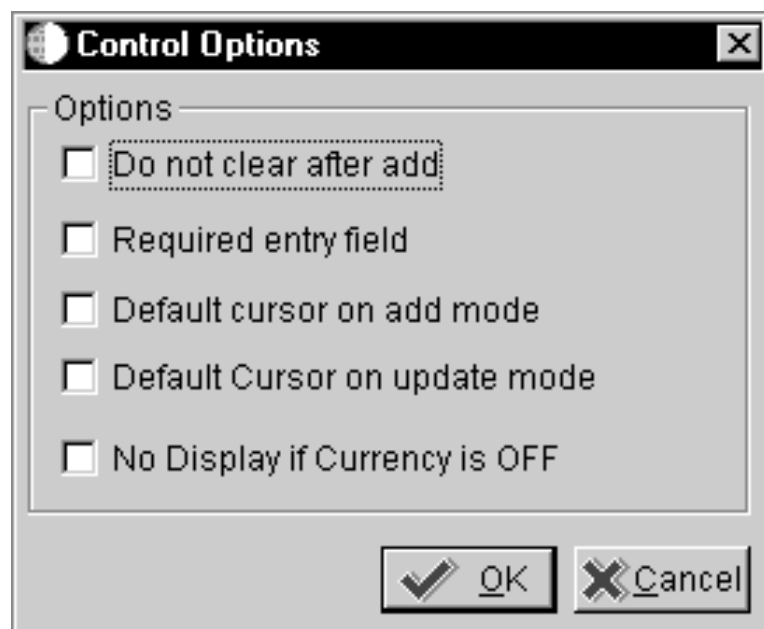
When you are designing an application you can decide whether to hide or show currency-sensitive controls, such as check boxes and radio buttons at runtime.

► **To show currency sensitive controls**

On Forms Design

1. Double click on the control you wish to display.
2. Click Options.
3. Verify that the No Display if Currency is Off option is turned off if you wish to display currency fields.

When currency is off, currency-sensitive controls are not displayed. If this option is turned off, currency fields are visible.



You must exit your current OneWorld session and re-enter in order to apply Currency Conversion changes.

Creating a Currency Conversion Trigger

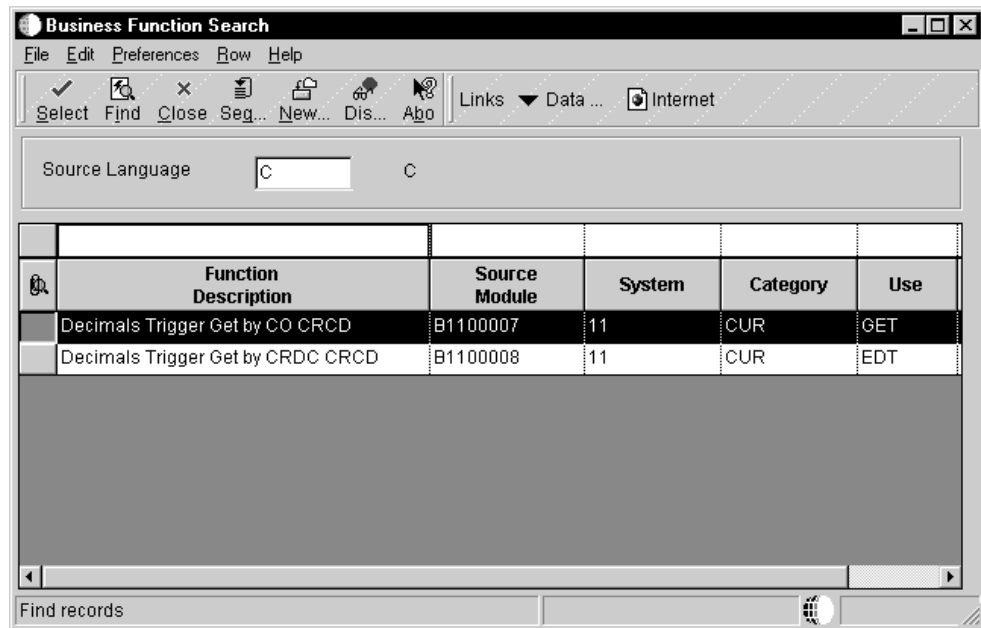
► **To create a currency conversion trigger**

1. From Object Librarian, check out the table that you wish to attach event rules to.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1223214	GL2463973	6/5/96	00:00:00	
INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
AEC-CONTRAC	APPL_PGF	3	1223214	DV5884811	6/24/98	15:07:29	
PETERSSON1	APPL_PGF	3	1223214	SP5811785	8/4/98	13:37:45	
SAYLERB1	APPL_PGF	3	1223214	BS5625464	8/4/98	18:57:09	
ULMERK	APPL_PGF	3	1223214	KU5729167	8/3/98	15:01:42	

- On Table Design Aid, choose Table ER from the Form menu.

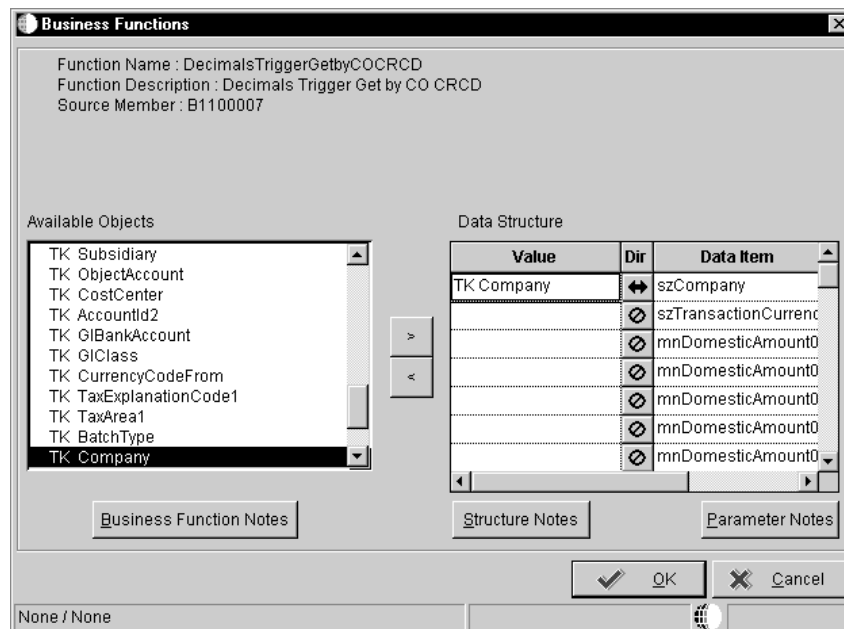
- On Event Rules Design, choose the *Currency Conversion* event and attach the currency trigger that you want to use.
- Click the Business Functions button. The Business Function Search form appears.



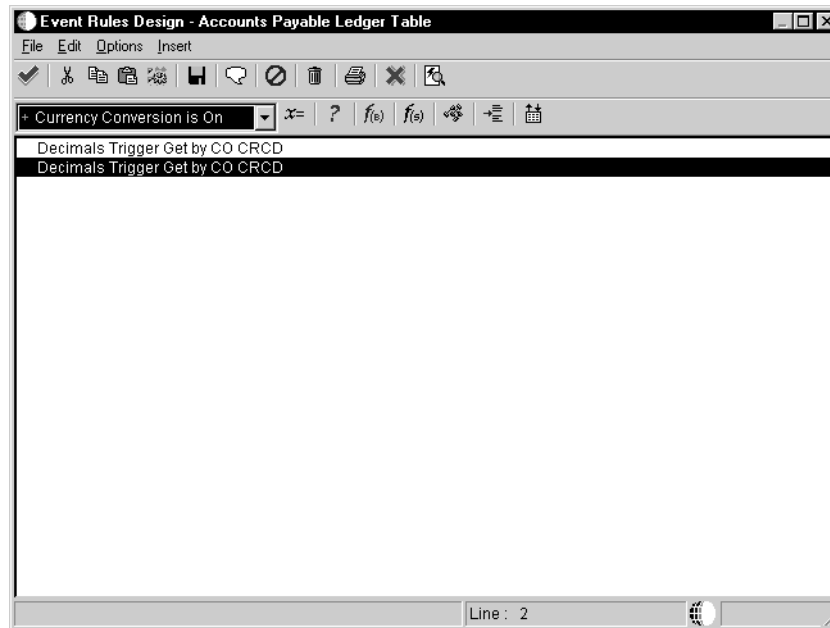
Use the QBE line to quickly search for selected business functions. You can use Category CUR or System Code 11 to find existing currency business functions. To read notes that describe the purpose of the business function, its parameters and program requirements, click the Attachments button.

5. Attach the table columns to the business function structure.

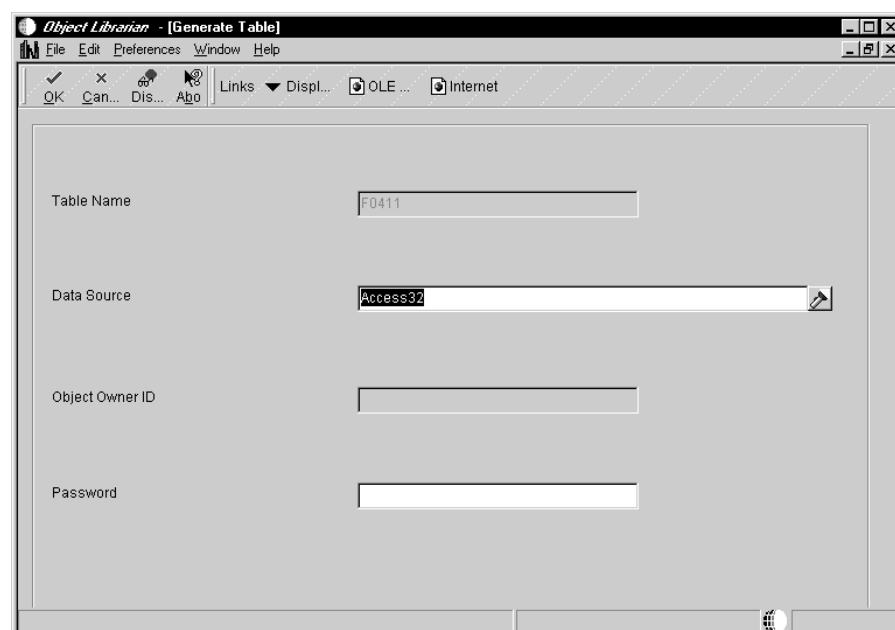
The available objects that appear are for table column only.



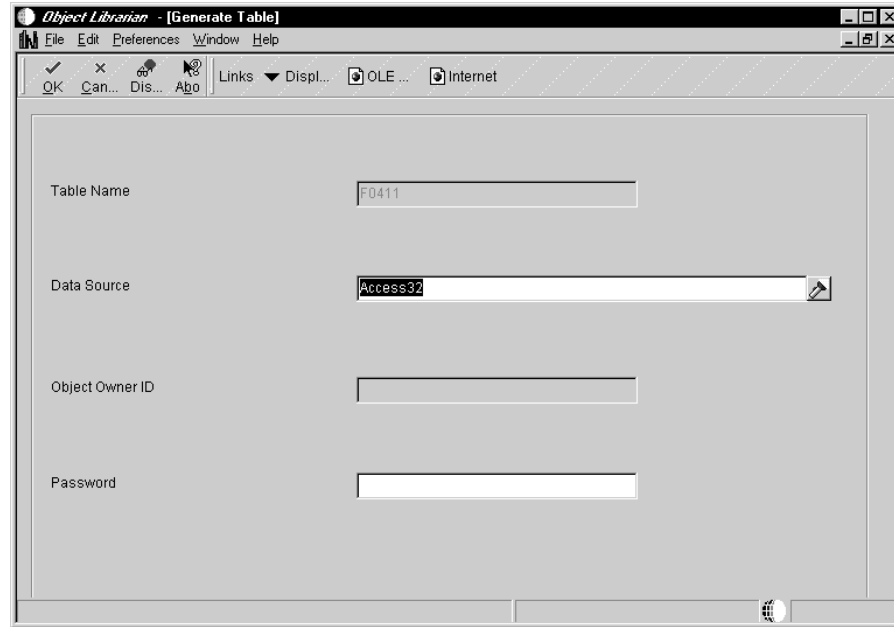
6. On Event Rules Design click Save, then click OK.



7. On Table Design Aid, choose Generate Table from the Form menu.



8. On Generate Table, choose the Data Source.

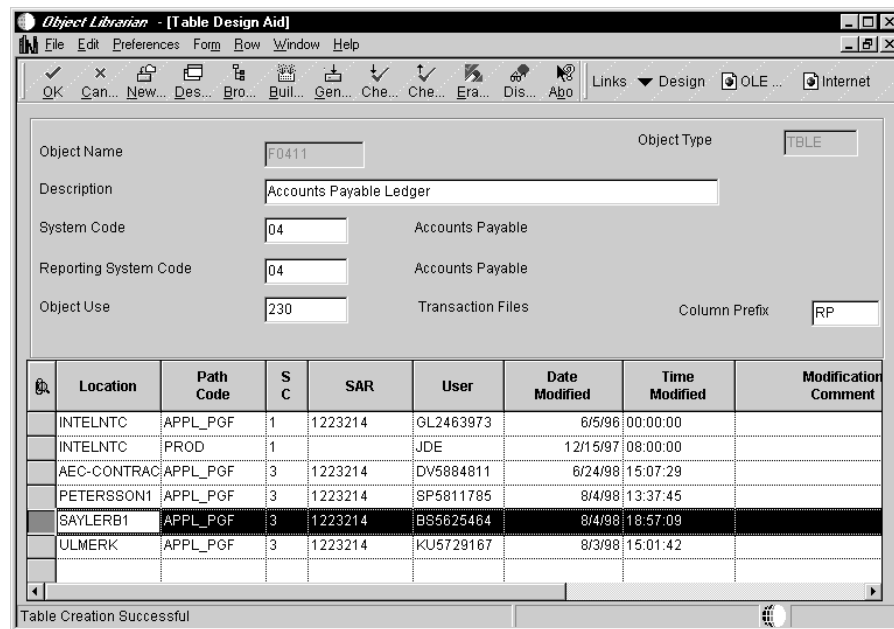


The dialog box 'Object Librarian - [Generate Table]' contains the following fields:

- Table Name: F0411
- Data Source: Access32
- Object Owner ID: (empty)
- Password: (empty)

9. On Table Design Aid, choose Build Triggers from the Form menu.

The Business Function Builder form appears.



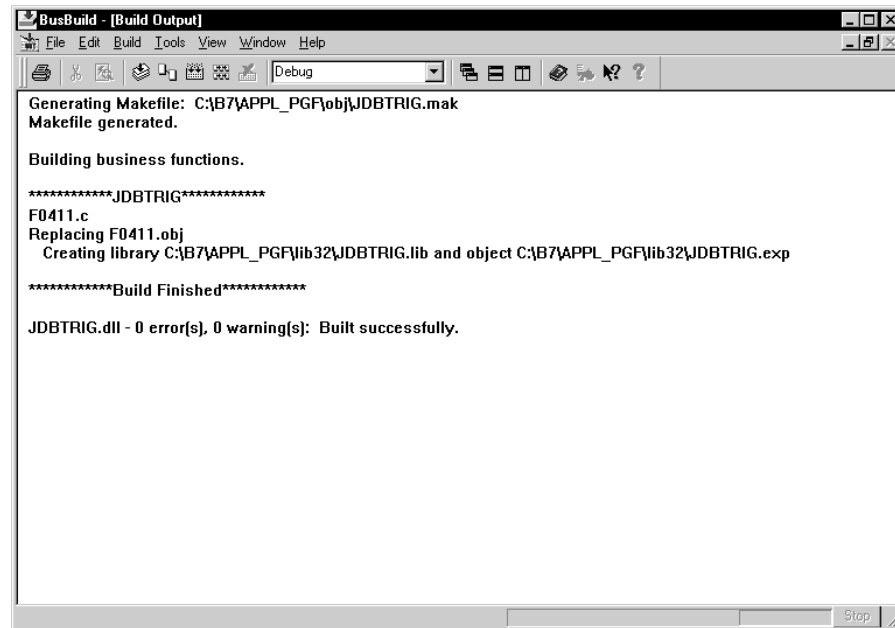
The dialog box 'Object Librarian - [Table Design Aid]' contains the following fields and a table:

- Object Name: F0411
- Object Type: TBLE
- Description: Accounts Payable Ledger
- System Code: 04 Accounts Payable
- Reporting System Code: 04 Accounts Payable
- Object Use: 230 Transaction Files
- Column Prefix: RP

	Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
	INTELNTC	APPL_PGF	1	1223214	GL2463973	6/5/96	00:00:00	
	INTELNTC	PROD	1		JDE	12/15/97	08:00:00	
	AEC-CONTRAC	APPL_PGF	3	1223214	DV5884811	6/24/98	15:07:29	
	PETERSSON1	APPL_PGF	3	1223214	SP5811785	8/4/98	13:37:45	
	SAYLERB1	APPL_PGF	3	1223214	BS5625464	8/4/98	18:57:09	
	ULMERK	APPL_PGF	3	1223214	KU5729167	8/3/98	15:01:42	

Table Creation Successful

The creation of the table event rule is complete. The newly created or modified table event rule functions are now called from the database APIs whenever the corresponding event occurs against the table.





Menu Design

You use Menu Design (P0082) to create, change, delete, copy, and filter menus and menu selections. Menu Design assists you in the following:

- Managing menus and menu selections
- Managing text overrides
- Defining runtime messages for menu selections
- Indicating the consequences of using particular menu selections
- Copying menu selections

In this section you will learn how to:

- ☐ Working with menus
- ☐ Working with menu selections
- ☐ Working with menu selection revisions

Understanding Menus

A menu is the entry point for running reports and applications. The Menu Master (F0082) table stores the following information, which identifies and characterizes the menu:

- Identifying information (ID and related system code)
- Level of detail
- Menu classification
- Menu Text Override (F0083)

Menu Filtering

OneWorld automatically filters menus based on your user ID so that only menu selections that apply to your job appear on your workstation. This feature allows you to maintain one set of menus (one database) with hundreds of menu selections, but you see only those menus that apply to your job.



Menus are filtered so the following do not appear:

- Menu selections that you or other users do not have authority to access. For example, Employee Information in Human Resources Management.
- Menu selections that are country-code-specific. If your user profile country code matches the country code for that menu, then the selection displays. For example, menu selections that pertain only to Canadian users, such as Canadian tax-related selections, appear only to Canadian users.
- WorldVision menu selections that are not installed on your workstation. For example, if WorldVision (the J.D. Edwards AS/400 product) is not installed on your workstation, that selection does not appear on the menu. You can distinguish a WorldVision menu selection from a OneWorld menu selection by looking at the Job to Execute number. A WorldVision Job to Execute number begins with a J, for example, J3413.

Menu Design Tables

Menu Design stores information in the following tables:

Menu Master (F0082)	Defines all menus but not the selections.
Menu Selection (F00821)	Contains the type of selection to be executed, selection consequences, and version information.
Menu Text Override (F0083)	Contains menu selection descriptions.
Menu Path (F0084)	Contains the menu selection icons.

Working with Menus

Menus are the entry point to J.D. Edwards applications and reports. To access an application or report from a menu, the application or report must be attached to a menu selection on the menu.

This chapter describes the following:

- ☐ Defining a new menu
- ☐ Reviewing selections for a menu
- ☐ Printing a menu report

Defining a New Menu

You can define a menu to include selections that enable you to access the applications and reports you need from one location.

To define a new menu

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, click Add.

3. On Menu Header Revisions, complete the following fields:

- Menu Identification
- Menu Title
- System Code
- Menu Classification
- Level of Display
- Advanced & Technical Operations
- Setup Menu

Field	Explanation								
Menu Identification	The menu name, which can include up to nine characters. J.D. Edwards standards are: • Menu numbers are preceded with a G prefix. • The two characters following the prefix are the system code. • The next characters further identify the menu. • The 4th character specifies a specific skill level. • The 5th character distinguishes two menus of the same system with the same skill level. For example, the menu identification G0911 specifies the following: <table> <tr> <td>G</td><td>Prefix</td></tr> <tr> <td>09</td><td>System code</td></tr> <tr> <td>1</td><td>Display level/skill level</td></tr> <tr> <td>1</td><td>First menu</td></tr> </table>	G	Prefix	09	System code	1	Display level/skill level	1	First menu
G	Prefix								
09	System code								
1	Display level/skill level								
1	First menu								

Field	Explanation																
Menu Title	Contains menu titles and menu selection descriptions.																
Menu Classification	The menu classification indicates the type of a menu. For example: a JDE Master menu or Company Master menu.																
Level of Display	The Level of Display field contains a number or letter identifying the level at which menus and processing options are displayed. The levels of display are as follows: <table> <tr> <td>A</td><td>Product Groups (for example, Job Cost, Manufacturing)</td></tr> <tr> <td>B</td><td>Major Products (for example, GL, AP)</td></tr> <tr> <td>1</td><td>Basic Operations</td></tr> <tr> <td>2</td><td>Intermediate Operations</td></tr> <tr> <td>3</td><td>Advanced Operations</td></tr> <tr> <td>4</td><td>Computer Operations</td></tr> <tr> <td>5</td><td>Programmers</td></tr> <tr> <td>6</td><td>Senior Programmers.</td></tr> </table>	A	Product Groups (for example, Job Cost, Manufacturing)	B	Major Products (for example, GL, AP)	1	Basic Operations	2	Intermediate Operations	3	Advanced Operations	4	Computer Operations	5	Programmers	6	Senior Programmers.
A	Product Groups (for example, Job Cost, Manufacturing)																
B	Major Products (for example, GL, AP)																
1	Basic Operations																
2	Intermediate Operations																
3	Advanced Operations																
4	Computer Operations																
5	Programmers																
6	Senior Programmers.																
Advanced & Technical Operations	For World: The advanced operations key is used to direct the menu selection '27' (Advanced Operations) to the appropriate menu. This menu designation must be preceded with an asterisk (*). For example, the General Accounting Advanced Operations menu would be *A093. <i>Form-specific information</i> For OneWorld: Each menu may optionally have an Advanced & Technical Operations menu to which it is associated and it is displayed as the last menu selection. This is normally used to categorize more advanced tasks.																
Setup Menu	For World: The technical operations control key is used to direct the menu selection '29' (Technical Operations) to the appropriate menu. This menu designation must be preceded with an asterisk (*). For example, the General Accounting Technical Operations Menu would be *A094. <i>Form-specific information</i> For OneWorld: The menu you enter in this field is associated with the menu item description: Setup Menu. This menu is automatically displayed at the bottom of the menu you specify in the Menu Identification field.																

Reviewing Selections for a Menu

You can review the selections included on a specific menu. You can accomplish this from the Work With Menus form or from the Menu Header Revisions form.

► To review selections for a menu

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, locate and choose a menu you wish to review.

Selection Number	Description	Jargon Code	Selection Consequences	Selection Type	Job to Execute	Version
1	SYSTEM CONTROLS & CODES		2	4		
2	Address Book Constants	01	2	2	P0000	
3	Country Constants Revisions	00	2	2	P0070	
5	Address Book Category Codes		2	1		
6	Who's Who Category Codes		2	1		
13	OTHER USER DEFINED CODES		2	4		
14	Search Types	98	2	2	P0004A	ZJDE0064
15	Standard Industry Codes	98	2	2	P0004A	ZJDE0065
16	Flash Messages	98	2	2	P0004A	ZJDE0066
17	State/Province Codes	98	2	2	P0004A	ZJDE0067
18	Country Codes	98	2	2	P0004A	ZJDE0068
19	Salutation Codes	98	2	2	P0004A	ZJDE0069

If you are working with the Menu Header Revisions form, choose Selections from the Form menu.

In either instance, the Work With Menu Selections form appears, from which you can view and edit selections for a specific menu.

Printing a Menu Report

You can print a report that lists the menus. You can print menus only or menus and menu selections. To print a menu report, choose Menu Print from the Form menu.

Example: Print Menus Report

R0082P

J.D. Edwards & Company

4/25/97 13:33:23

Print Menus

Page - 1

Menu	Description	SY	LOD	Class				
DEMO	APPLICATIONS	00						
Sel #	Description	Job To	Form	Version	Ctry	Appl	Run Time	SC
		Execute	Name			Ovrld	Message	
0	APPLICATIONS							1
1	Journal Entries	P0911		ZJDE0001				3
2	General Journal Posting	R09801						3
3	Company Numbers and Names	P0010						3
4	Budget vs.Actual Comparison	P09210		ZJDE0001				2
5	Account Ledger Inquiry	P0911L						1
6	Original Budget Update	P14102		ZJDE0001				3
7	Online Consolidations	P09218		ZJDE0001				1
8	Manufacturing Variance Inquiry	P3102		ZJDE0001				1
11	Bill of Material	P3002		ZJDE0001				1
12	Routings	P3003		ZJDE0001				1
13	Forecasting	P3460		ZJDE0001				1
14	Customer Service	P4210		ZJDE0001				1
15	Planners Workbench	P3401		ZJDE0003				1
16	Schedulers Workbench	P31225		ZJDE0001				1

Working with Menu Selections

Work With Menu Selections displays available selections for the selected menu. Use Work With Menu Selections when you are:

- ☐ Adding or changing a menu selection
- ☐ Adding an application to a menu
- ☐ Adding or changing Web addresses on OneWorld Explorer Help
- ☐ Creating a Web view subheading on a menu
- ☐ Linking menus
- ☐ Creating fast path selections

Adding or Changing a Menu Selection

To add a menu selection for an application or report, you must first name the menu selection by assigning a description and unique selection number before you can add a menu selection for an application or report to the menu.

After naming a menu selection, indicate the selection type and define it.

- Selection Type specifies the type of program that is executed for the menu selection. You use OneWorld Application, OneWorld Report, World Vision, or Windows Application to execute a specific application, report, or program.
- The Subheading selection type does not perform an action. You use it to logically group menu selections on the menu. Subheading selections appear on the menu only in Web view.
- You use the Menu selection type to call another menu.

Adding or changing a menu selection consists of the following:

- Naming the menu selection
- Defining the menu selection

▶ To name a menu selection

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection you wish to name.

Selection Number	Description	Jargon Code	Selection Consequences	Selection Type	Job to Execute	Version
1	SYSTEM CONTROLS & CODES		2	4		
2	Address Book Constants	01	2	2	P0000	
3	Country Constants Revisions	00	2	2	P0070	
5	Address Book Category Codes		2	1		
6	Who's Who Category Codes		2	1		
13	OTHER USER DEFINED CODES		2	4		
14	Search Types	98	2	2	P0004A	ZJDE0064
15	Standard Industry Codes	98	2	2	P0004A	ZJDE0065
16	Flash Messages	98	2	2	P0004A	ZJDE0066
17	State/Province Codes	98	2	2	P0004A	ZJDE0067
18	Country Codes	98	2	2	P0004A	ZJDE0068
19	Salutation Codes	98	2	2	P0004A	ZJDE0069

3. On Work With Menu Selections, click Add or choose an existing selection to change.

Menu Design - [Menu Selection Revisions]

File Edit Preferences Form Window Help

OK Cancel Dis... Add Links Define Internet

Selection Number

Selection Description

Selection Consequences

Jargon Code

Country Code

Selection Type

☒ OneWorld Application ☐ OneWorld Report ☐ World Vision

☐ Windows Application ☐ Web View Sub-Heading ☐ Menu

Icon

4. On Menu Selection Revisions, complete the following required fields:

- Selection Number
- Selection Description
- Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

5. The following fields are optional, complete if necessary:

- Jargon
- Country Code

If the base language is a double-byte language, a Search Description field is shown below the Country Code field. Enter the single-byte search description to be used by Menu Word Search. Menu Word Search uses only single-byte search descriptions.

Field	Explanation
Selection Number	Used to determine the order of menu items and allow them to be selected by this number.
Selection Description	Contains menu titles and menu selection descriptions.
Selection Consequences	The selection consequences tell the user what the consequences are in taking that specific menu selection.

Field	Explanation
Jargon	A code used to designate the reporting system number for entering specific text or “jargon”. See User Defined Codes, system code '98', record type 'SY' for a list of valid values.
Country Code	The Menu Country/Region Codes field contains the region code (3 bytes) for all 24 menu selections for each menu record. This region code is used to mask those international selections that are country specific; i.e. 1099 processing in the US and VAT tax processing in Europe.

► To define the menu selection

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to define a menu selection.
3. On Work With Menu Selections, choose an existing selection to define.
4. On Menu Selection Revisions, indicate one of the following Selection Types:
 - OneWorld Application
 - OneWorld Report
 - World Vision
 - Windows Application
 - Sub-Heading
 - Menu
5. From the Form menu, choose Define.

A form that is specific to the selection type appears.

6. Define options for the selection type indicated.
7. Click OK.

What You Should Know About

Deleting a Menu

When you delete a menu, you delete the menu and any menu selections available on that menu. The applications called by the menu selections are not deleted, and you can access these applications from other menus.

Adding an Application to a Menu

You can add OneWorld applications and reports, WorldVision applications, and Windows applications to a menu. You can also link a menu to another menu.

Complete the following tasks:

- Add a OneWorld application to a menu
- Add a OneWorld report selection to a menu
- Add a WorldVision application to a menu
- Add a Windows application to a menu



To add a OneWorld application to a menu

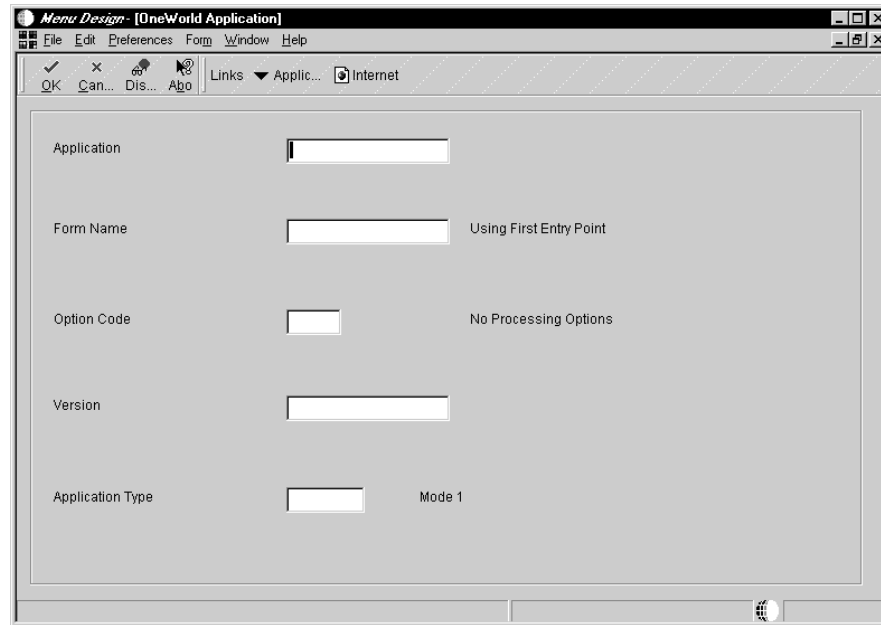
You can use this procedure to add a OneWorld application created in Forms Design as a menu selection.

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to add a OneWorld application.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following required fields:
 - Selection Number
 - Selection Description
 - Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

5. Click the OneWorld Application option, and from the Form menu, choose Define.



6. On OneWorld Application, complete the following fields:

- Application
- Form Name

You can use this field to define a specific entry point for the OneWorld application. If you leave this blank, the program's first entry point is used.

- Option Code
- Version
- Application Type

7. From the form menu, choose Application to view and choose from a list of available applications. Likewise, from the Form menu you can choose Versions to search for available versions.

Field	Explanation
Option Code	For World, this code specifies the function of a menu selection using the DREAM Writer when F18 is pressed. F18 may be locked out by simply replacing code 1 with 3 or code 2 with 4. This code, in conjunction with the version number and the option key, provide the following functions: Code 1 version — mandatory; option key — form i.d. F18 displays processing options. Selection = blind DREAM Writer execution. 2 version — blank; option key — form i.d. F18 displays DREAM Writer versions list. Selection = DREAM Writer versions list. 2 version — not blank; option key — form i.d. F18 displays DREAM Writer versions list. Selection = blind execution, batch. Review the HELP instructions for Menu Information (Menu Locks) (P0090) for a detailed explanation of codes related to job submission and control. For OneWorld, this code specifies whether the user will be prompted for additional information prior to running the application. Available values are: 0 No processing options 1 Blind execution (no prompt) 2 Prompt for version 3 Prompt for values
Version	Version identifies a specific set of data selection and sequencing settings for the application. Versions may be named using any combination of alpha and numeric characters. Versions that begin with 'XJDE' or 'ZJDE' are set up by J.D. Edwards.
Application Type	Complete with a user-defined, alphanumeric value. This field exists in the JDE user profile and within each menu and menu selection record. When security is active, the value of this field in the user profile is compared with the value in the corresponding menu lock. The values must be equal in the user profile and menu lock to access the menu. A blank in this field in the user profile gives the user all authority. A blank in this field in the menu record indicates no security exists on this menu.



To add a OneWorld report selection to a menu

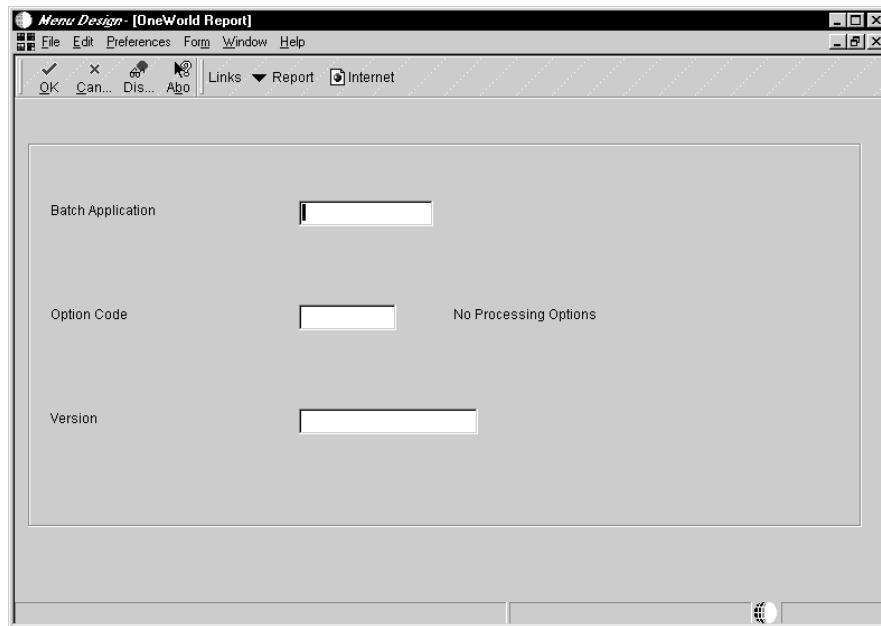
You can add a report created in the OneWorld Report Design tool as a menu selection.

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to add a OneWorld report selection.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following required fields:
 - Selection Number
 - Selection Description
 - Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

5. Click the OneWorld Report option, and from the Form menu, choose Define.



The screenshot shows a dialog box titled "Menu Design - [OneWorld Report]". It has a menu bar with "File", "Edit", "Preferences", "Form", "Window", and "Help". Below the menu bar is a toolbar with icons for "OK", "Cancel", "Dismiss", and "Apply", followed by buttons for "Links", "Report", and "Internet". The main area of the dialog contains three input fields: "Batch Application", "Option Code", and "Version". To the right of the "Option Code" field, the text "No Processing Options" is displayed. The dialog box has a standard Windows-style border with minimize, maximize, and close buttons in the top right corner.

6. On OneWorld Report, complete the following fields:
 - Batch Application
 - Option Code
 - Version

Use Form menu options to view and choose from a list of reports and versions.

► **To add a WorldVision application to a menu**

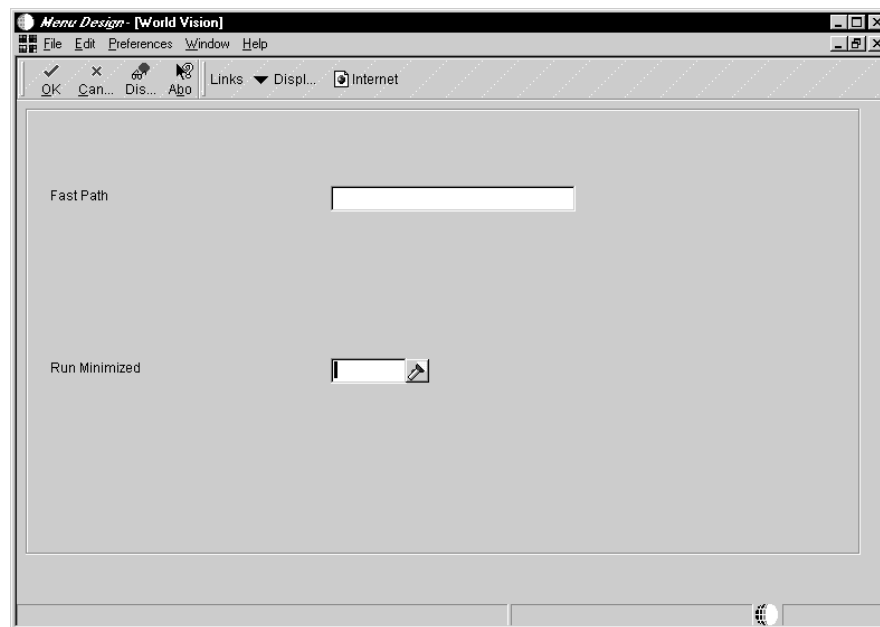
You can add a WorldVision application as a menu selection.

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to add a WorldVision selection.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following required fields:
 - Selection Number
 - Selection Description
 - Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

5. Click the WorldVision option, and from the Form menu, choose Define.



6. On WorldVision, complete the following fields:
 - Fast Path
 - Run Minimized

Field	Explanation
Fast Path	The path field contains the path used for client based menus. The path describes where the application is located on your computer or network. A path includes the drive, folders, and subfolders that contain the application to be executed. To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.
Run Minimized	The Run Minimized flag determines if a Windows application is to be minimized to an icon when you open it.

To add a Windows application to a menu

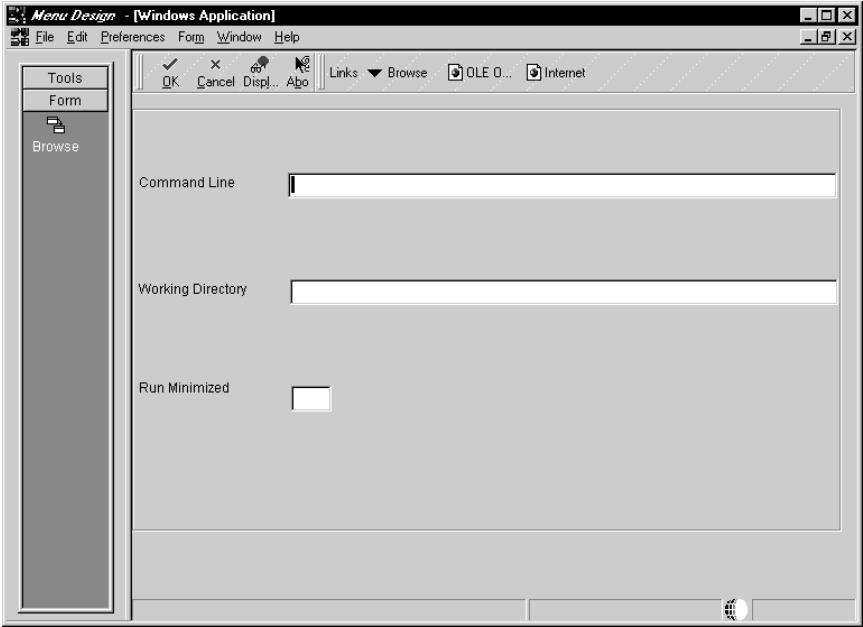
You can add any Windows applications as menu selections. For example, you can add the Windows programs such as Calendar, Clock, Note Pad, or Write.

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to add a Windows application.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following required fields:
 - Selection Number
 - Selection Description
 - Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

5. Click the Windows Application option, and from the Form menu, choose Define.



6. On Windows Application, complete the following fields, or click the Browse button to search for the Windows application:

- Command Line

Enter the executable of the Windows application you want to add to the OneWorld menu, such as winword.exe.

- Working Directory

Enter the path on your local machine where the Windows application resides.

- Run Minimized

Field	Explanation
Command Line	The path field contains the path used for client based menus. The path describes where the application is located on your computer or network. A path includes the drive, folders, and subfolders that contain the application to be executed. To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.
Working Directory	The path field contains the path used for client based menus. The path describes where the application is located on your computer or network. A path includes the drive, folders, and subfolders that contain the application to be executed. To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.

Adding or Changing Web Addresses on OneWorld Explorer Help

You can add Web addresses or change some of the addresses that appear on the Help menu of OneWorld Explorer. From the Help menu on OneWorld Explorer, there is an option called “J.D. Edwards on the Web.” From this option, a list of Web addresses appear, and there is a line that separates the addresses. The addresses above this line, which include J.D. Edwards Home Page and Contact Us, are hard-coded into OneWorld, which means you cannot change them. You can, however, change the Web addresses below the line, or add your own Web addresses to the list.

► To add or change Web addresses on OneWorld Explorer Help

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, in the Menu ID query-by-example field, type HELP, then click Find.

The Web Access menu appears.

3. Choose the Web Access menu and click Select.
4. On Work With Menu Selections, click Add to add a new Web address, or choose a row and click Select to change an existing Web address.
5. On Menu Selection Revisions, complete the following required fields:
 - Selection Number
 - Selection Description
 - Selection Consequences

If you chose an existing selection to change, the Selection Number field is disabled.

6. Click the Windows Application option, and from the Form menu, choose Define.
7. On Windows Application, complete the following fields:
 - Command Line

Enter the Web address that you want to add to the Help menu, such as <http://www.jdedwards.com>.

- Working Directory

Because you are entering a Web address, you do not need to complete this field.

- Run Minimized

Creating a Web View Subheading on a Menu

Use Web subheadings to logically group menu selections on the menu. Subheadings appear on the menu in Web view only and do not perform an action.

To create a Web view subheading selection

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to create a subheading.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following field:
 - Selection Number
5. Click the Web View Sub-Heading option.
6. Click OK to complete the Web view subheading assignment.

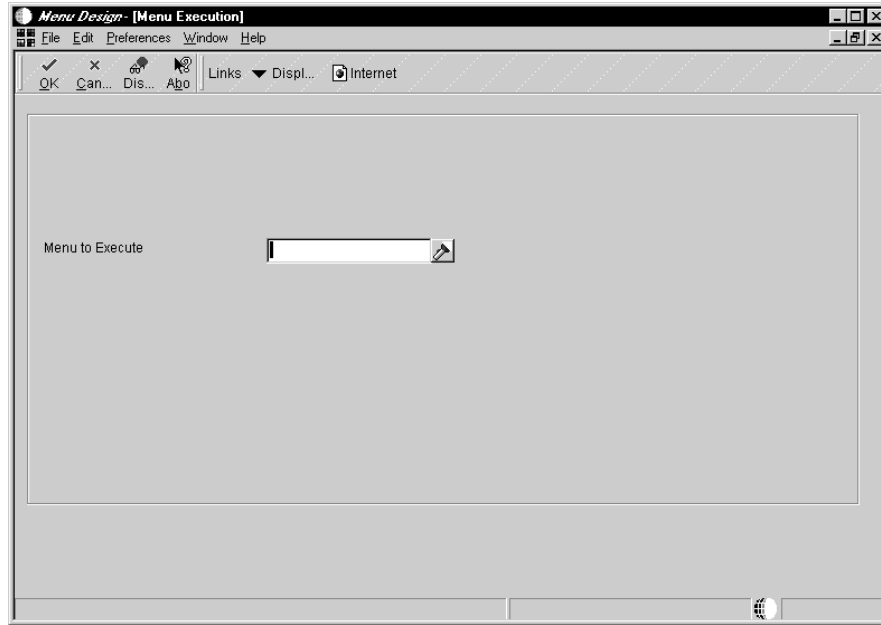
Linking Menus

You can add a menu selection that displays another menu.

To link another menu to a menu

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. Find and choose the menu on which you wish to add a selection.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, click the Menu option.
5. From the Form menu, choose Define.



6. On Menu Execution, complete the following field or click the visual assist to search for menus:
 - Menu to Execute

Field	Explanation
Menu to Execute	The specific menu to be executed as a selection on a menu.

Creating Fast Path Selections

You can quickly move among menus and applications by using fast path commands. A fast path command is:

- An abbreviation that is either shipped with J.D. Edwards demo data or which you define to suit your business environment. For example, the fast path OL takes you to the application Object Librarian so you can work with OneWorld objects.
- A combination of menu selection and menu number. For example 2/G01 (menu selection number 2 on menu number G01) takes you to Work With Addresses in Address Book. As you become more familiar with OneWorld menu abbreviations, you might find fast path a quicker way to navigate to an application.

You can set up your own a fast path abbreviations to access frequently-used applications.

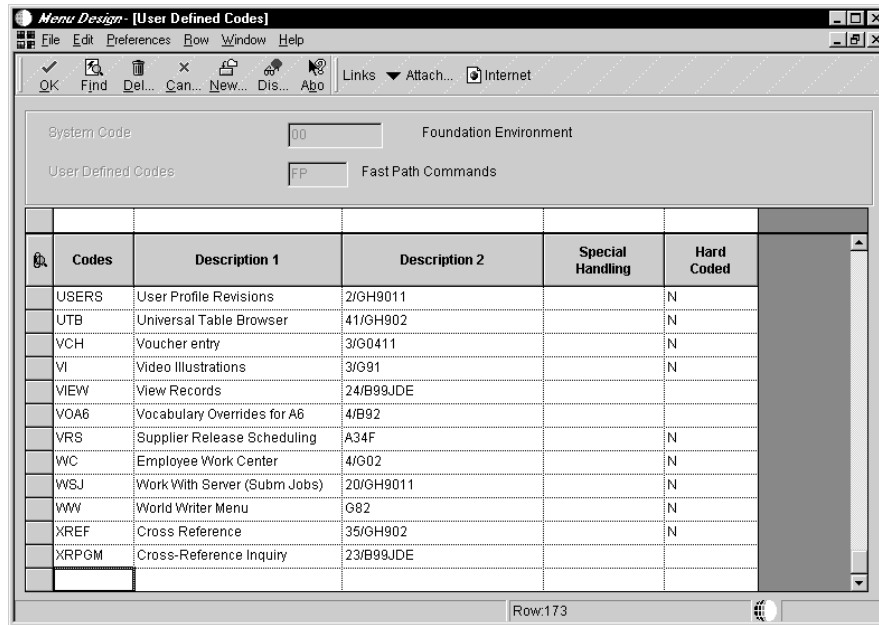
► **To create a fast path selection**

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection for which you wish to create a fast path.
3. On Work With Menu Selections, from the Form menu, choose Fast Path Revs.

Codes	Description 01	Description 02	Special Handling	Hard Coded
	Call Group Job Window	CALL P98GRP		N
1K	Address Book Constants	2/G0141		N
3K	AVR Constants	2/G03B41		N
4K	AVP Constants	2/G0441		N
9K	G/L Constants	2/G0941		N
AAI	Automatic Accounting Instrucs.	15/G00		N
AAIT	AAI Translations	06/G0941		N
ACCT	Single Account Revision	16/G09411		N
ACON	Advanced Configurator	G3231		N
APD	Advanced PDM	G3031		N
ASF	Advanced Shop Floor Control	G3131		N
ASI	Application Specific Instrucs.	7/G97PTF		N
ASIE	Application Specific Instrucs.	1/B97ASI		N

4. On Work With User Defined Codes, click Add.



5. On User Defined Codes, click inside the grid, then press the Ctrl and End keys to display the bottom of the grid.
6. To add a user defined code for a new fast path, complete the following required fields in the last row of the grid:
 - Codes
 - Description 1
 - Description 2

You enter the abbreviation for the fast path in the Code field. Enter the description of the abbreviation, such as the name of the menu selection, in the Description 01 field. Enter the selection number and menu number in the Description 02 field.

To determine the selection number for the fast path you created (for example, selection number 2 on menu G01), use Work With Menu Selections. Do not count the menu selections in OneWorld Explorer because the menu might be filtered.

Working with Menu Selection Revisions

You can revise your existing menu selections to change menu text for languages, change menu selection text, or renumber a menu selection. This chapter describes the following tasks:

- ☐ Copying a menu selection
- ☐ Changing menu text for languages
- ☐ Changing menu selection text
- ☐ Renumbering a menu selection

Copying a Menu Selection

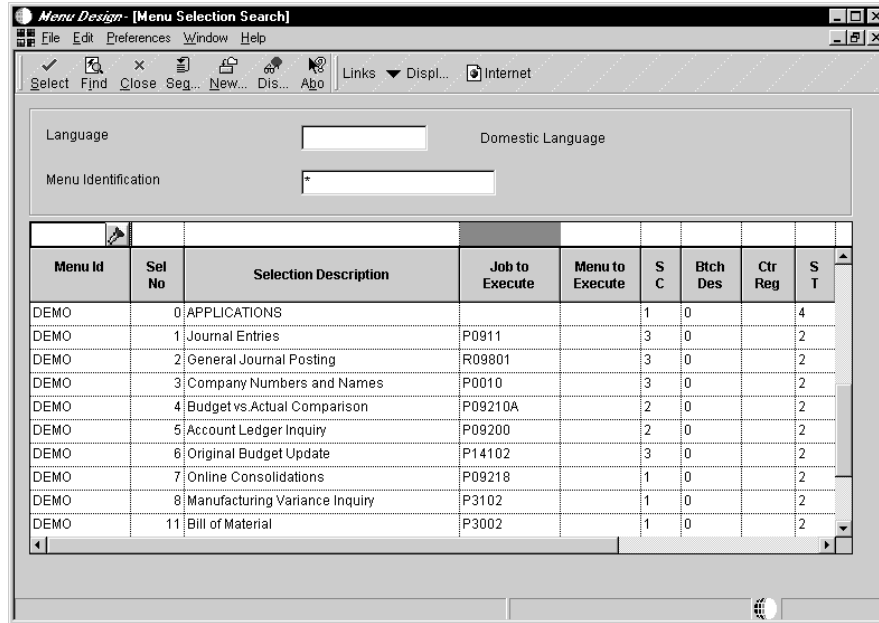
You can copy an existing menu selection and attach it to another menu.



To copy a menu selection

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection you wish to copy.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, enter the new selection number and choose Copy from the Form menu.



- On Menu Selection Search, choose an existing menu selection.

The selection description, consequences, and type are inserted into the newly added menu selection.

Changing Menu Text for Languages

You use Title Overrides to change the language and description of a menu selection.

► To change menu text for languages

On System Administration Tools (GH9011)

- Choose Menu Design (P0082).
- On Work With Menus, find and choose the menu for which you wish to change menu text.
- On Work With Menus, choose Header from the Row menu selection.

Menu Design - [Menu Header Revisions]

File Edit Preferences Form Window Help

OK Cancel Dis... Abo Links Select... Previo... Next Internet

Menu Identification: G0141

Menu Title: Address Book Setup

System Code: 01 Address Book

Menu Classification: 1 JDE Master

Level of Display: 4 Setup Operations

Advanced Technical Operations:

Setup Menu:

Work With Menus Menu Header Revisions

4. On Menu Header Revisions, from the Form menu, choose Title Overrides.

Menu Design - [Menu Text Overrides]

File Edit Preferences Window Help

OK Del... Can... New... Dis... Abo Links Displ... Internet

Menu Identification: G0141

Address Book Setup

Language	Menu Text

Row:1

5. On Menu Text Overrides, enter the desired language code (such as S for Spanish) and text description (such as the Spanish description of the menu selection) in the following fields and click OK.
 - Language
 - Menu Text

Field	Explanation
Language	A user defined code (system 01/type LP) that specifies a language to use in forms and printed reports. Before any translations can become effective, a language code must exist at either the system level or in your user preferences.
Menu Text	Contains menu titles and menu selection descriptions.

Changing Menu Selection Text

You use Text Overrides to change a menu selection's text.

To change menu selection text

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu for which you wish to change menu's selection text.
3. On Work With Menu Selections, choose the menu selection you want to change and click Select.
4. On Menu Selection Revisions, choose Text Overrides from the Form menu.
5. On Menu Text Overrides, complete the following fields and click OK:
 - Language
 - Menu Text

Changes in menu text are not displayed until the changed menu is closed and reopened. You can also choose Refresh from the View menu to update the menu text.

Renumbering a Menu Selection

You can renumber menu selections from both Work With Menu Selections and Menu Selection Revisions. You can edit each selection to change the sequence of selections on a menu. You cannot rearrange menu selections by clicking and dragging them.

To renumber a menu selection

On System Administration Tools (GH9011)

1. Choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu for which you wish to change menu's selection number.
3. On Work With Menu Selections, choose the menu selection you want to change and click Select.
4. On Menu Selection Revisions, from the Form menu, choose Renumber.

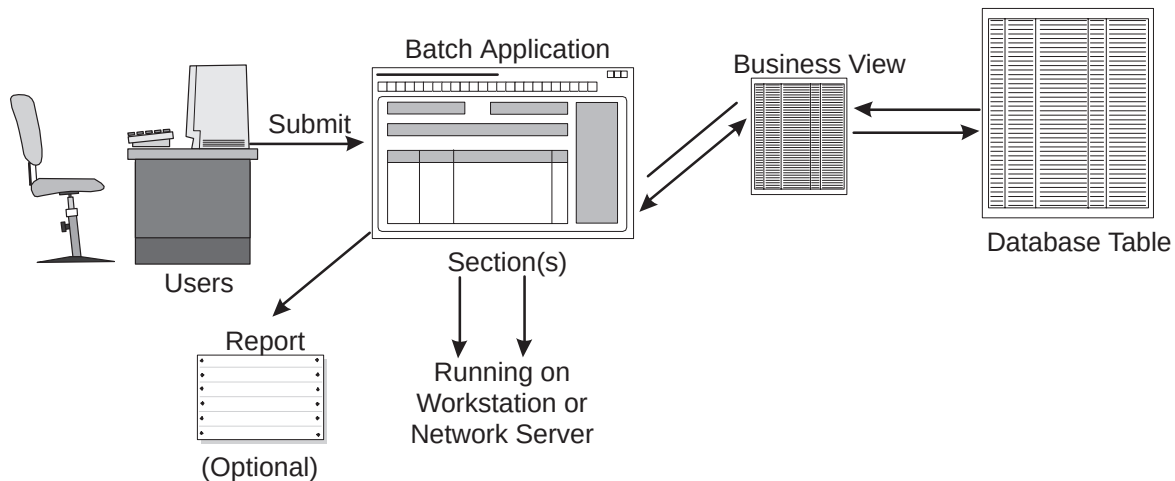
5. Complete the following field, and click OK.
 - New Selection Number

Field	Explanation
New Selection Number	Used to determine the order of menu items and allow them to be selected by this number.

Batch Processing

Batch Processing

A batch process is an application that processes automatically without user interaction. The following graphic illustrates the flow of a batch process.



With interactive processes, the engine pauses at certain points and waits for the user's interaction or response. Based on the user's response, the program decides what to do next. There is no user interaction with batch processes. The engine executes the logic attached to the events within the report or process without any user interaction. Once a batch process is launched, the user has no control over the flow of the logic within the batch process. If you need to change the flow of the logic within the process or report, you make those changes using Report Design.

Examples of batch processes include reports, subsystem jobs, database output, and table conversion.

Subsystem jobs are batch processes that constantly run in the background and offload processor resources. You can also use subsystem jobs to move activities along in a process, such as an escalation process in Workflow, which moves unanswered messages from one user to another after a certain period of time.

The database output function within Report Design allows you to update or insert records within tables. Table conversion transfers data from one table to one or more tables, changes the data or schema of a OneWorld table, and transfers data from a single OneWorld business view to one or more tables.

This section describes some of the advanced features behind batch and report processing and is intended to give you an understanding of how the features of batch and report processes work.



The following topics are discussed:

- ☐ Understanding batch processing
- ☐ Understanding section processing
- ☐ Understanding custom sections
- ☐ Understanding level-break processing
- ☐ Understanding batch events
- ☐ Understanding batch runtime processing

Understanding Batch Processing

You create reports and batch processes using the Report Design tool, and you use OneWorld Explorer to access the reports. You can also associate individual reports and batch processes with applicable menus. One difference between a batch process and a report is that a batch process does not usually print a report. You can use batch processes to update tables. You can also create a batch process that prints a report showing the results of the batch process. OneWorld Report Design also uses configurable directors. You can use a preconfigured director, for example for Financials, to step you through the creation of a report.

OneWorld reports contain all the specifications for the report, including section and field layout, business views, event rules, data selection, sequencing, and database output. Reports are named with a unique identifier.

Each report contains one or more *sections*, which you create in Report Design. Sections are self-contained elements that are used as building blocks to construct a report. You can relate sections to one another or use them as standalone reports. Sections can also serve a special purpose, such as headers and footers.

The following is the typical process that occurs when you create a report or batch process.

Report Sections

OneWorld sections include headers, footers, and body sections. You can use three types of body sections:

Group

You use group sections when you want to display data in either a vertical or horizontal format. Group sections display row headings each time a new record is read. You can use free form groups based on the process to be performed.

Columnar

Columnar sections list data in a columnar fashion, with column headings at the top of each report page and rows of data beneath each heading. They use a query-like format. Column headings are not repeated when a new record is read.

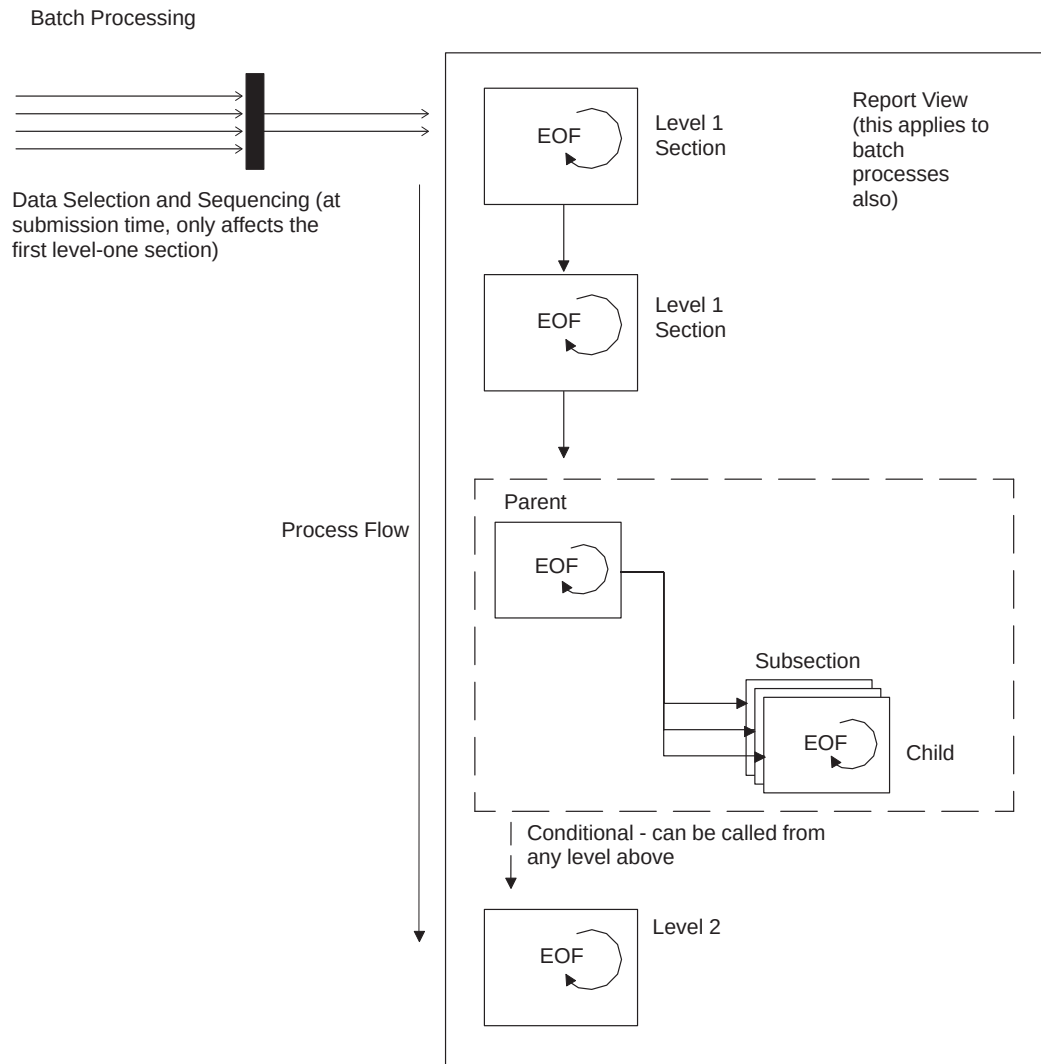
Tabular

Tabular sections present report data in columns, rows, and cells. You use tabular sections when you need to control the information for each item, such as information in financial reports. Tabular sections print information only when there is a level break. Tabular sections automatically sum and creates total at level break.

Group, columnar, and tabular sections are all level-one (independent) sections. Header and footer sections are level-two (dependent) sections.

Because each kind of body section within a report has certain properties and characteristics specific to it, you must determine which body section best satisfies your reporting needs.

The following graphic illustrates how the batch engine processes sections within a report.



The report view within Report Design determines the flow of the report. For example, if you have multiple level-one sections within your report, each having their own business view, the first level one section (such as V0101E) is executed before the next level-one section (such as V4211A). Any level-two sections that exist between the level-one sections are processed when they are called from a level-one section or are connected to one. If the report contains a joined section (parent/child section) after the level-one sections, the parent/child section will be processed independently of the first two sections. Child sections are in subsection joins.

The system reads and processes all records in a section from beginning to end within a section based on its data selection. If you do not specify data selection, then all the records in the table will be read until the end of the file (EOF).

The parent/child section in the middle of the graphic illustrates the flow of a joined (parent/child) section. Parent/child sections can be called at any point within the process flow. Depending on how your sections are joined (one-to-one, one-to-many, or many-to-many), records are fetched from the child business view for each corresponding parent record. When all parent records have been processed, the system

continues to the next section. If any database updates are performed in the first section over the V0101 business view, that change will be reflected when the records are fetched for the parent/child join section.

The last section shown in the graphic (the conditional section) can be called from any of the previous sections by using the system function `DO_Custom_Section`. A conditional section is considered a level-two section because you must call a conditional section from another section. Memory allocation occurs at the beginning of the section so you must hide and show objects in the section as needed because memory is allocated and freed once, instead of each time a section is called. You can place conditional logic in your `INIT_SECTION` or `END_SECTION` event to hide and show objects.

OneWorld processes all section types the same way, except for tabular sections, which output to the report only when the system encounters a level break. In tabular sections, the system does not initialize the `DO_Section` for each record. Instead, it summarizes the records to the lowest level-break level. Tabular sections are similar to level breaks in that the output to a report is similar to a level-break header section.

The flow of sections in a report or batch process depends on how you set up the sections in the report view in Report Design. In Report View, you can move sections up or down, which affects the sequence of execution for the sections, except for conditional sections, subsections, level-break sections, page-headers, page-footers, and report headers. The report execution takes place from top to bottom for sections, and top to bottom, left to right for objects within a section. In report processing, the system processes all level-one sections in the order in which they appear.

Independent (level-one) sections include:

- Group sections
- Columnar sections
- Tabular sections

Level-one sections can also be parent sections.

Dependent (level-two) sections include:

- Level-break headers
- Level-break footers
- Total sections (used in tabular reports)
- Subsections
- Custom sections

All level-two sections are processed as dependents of the level-one section. Custom sections appear as a level-one section in the report view, but are not processed unless explicitly called by the `DO_Custom_Section` system function.

See *Using a Custom Section* for more information about custom sections.

Sections in themselves can be considered “mini reports,” or batch processes, and each section can contain its own business view. Because other sections in your report might contain different business views, you must add any data sequencing and selection to the template for each section using Report Design.



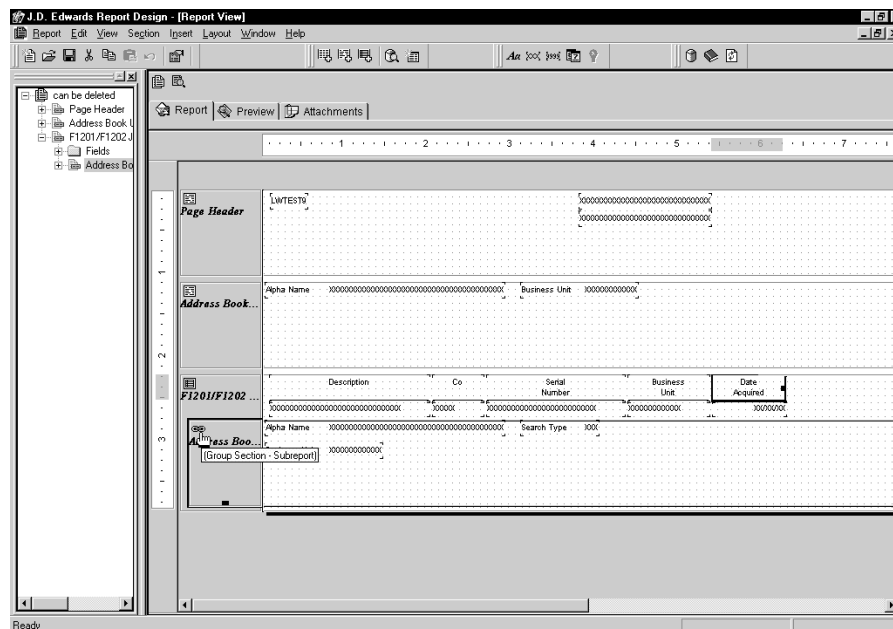
If a user sets up data sequencing and selection in the batch version at runtime, sequencing and selection will only affect the first section of the report or batch process. Therefore, when you design a report or batch process, consider the impact on the users because they will not be able to change data selection and sequencing for any other sections.

Even if another section in a report has the same business view as the first section, the other section will not adopt the first section’s data selection and sequencing. If you want a section to adopt another section’s data sequencing and selection, you can copy a section’s data sequencing and selection information using the Set User Selection system function in Event Rules.

Whether a level-one section has a business view attached to it or not, it will be executed at least once. This is important if you want to execute special event rule logic and you want to attach it to that section’s DO_Section event.

You can use the Report Design Director in Report Design to create sections. You can use the Director to build sections, or you can bypass the Director and build your section by choosing and modifying objects. You can also add sections using the Director or by choosing and modifying objects as needed.

As you create sections, you can view a representation of the sections you have created using the Report View tool, as shown in the following example:



The Report Tree View window on the left side displays the hierarchical structure of all sections in your report, including fields and data items used in each section. If you add a new section, it appears at the bottom of the hierarchy.

See Also

- The *Enterprise Report Writing Guide* for information about the fundamentals of creating reports using Report Design.
- *Subsystem Jobs* for information about how to create subsystem jobs.
- *Understanding Database Output* for more information about the database output function in Report Design.
- The *Table Conversion Guide* for information about how to create a table conversion using the table conversion tool.

Understanding Section Processing

When a batch process or report is processed certain events occur, whether you use group, columnar, or tabular sections. You can attach logic at these events. Your logic may be dependent on what happens before and after a particular event.

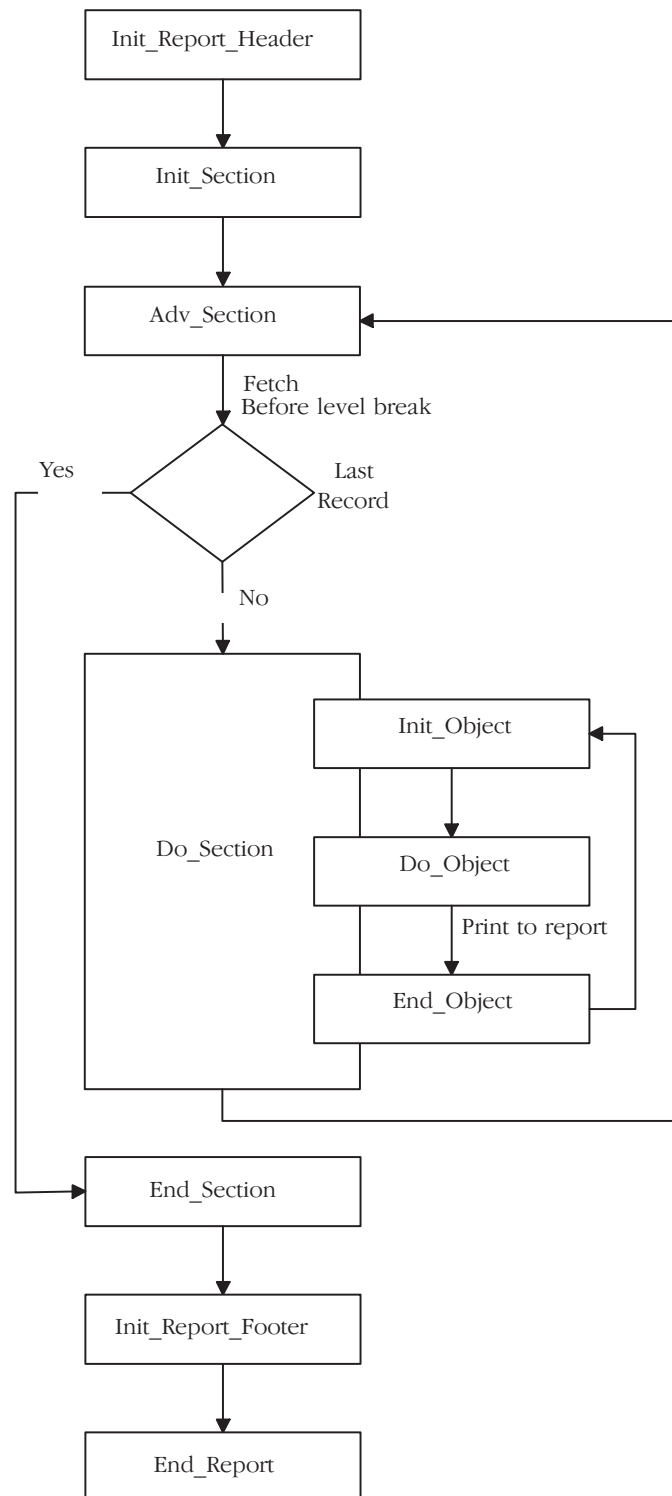
The report header is processed before the section is processed, and the level-one section is initialized by the INIT_Section event. After INIT_Section, the system processes the ADV_Section, Page Footer, Page Header, and so on. When the system finishes processing the first level-one section and all dependent sections associated with it, it repeats the process for the next level-one section.

The Do_Section event is discussed in *Understanding Batch Events*, and the level-break section is discussed in *Understanding Level-Break Processing*.

Processing Group and Columnar Sections

Group and columnar sections are alike in their processing because they both write to the output after each record is read, unlike tabular sections, which write to the output only when there is a level break. This output is determined by the data sequencing of that section.

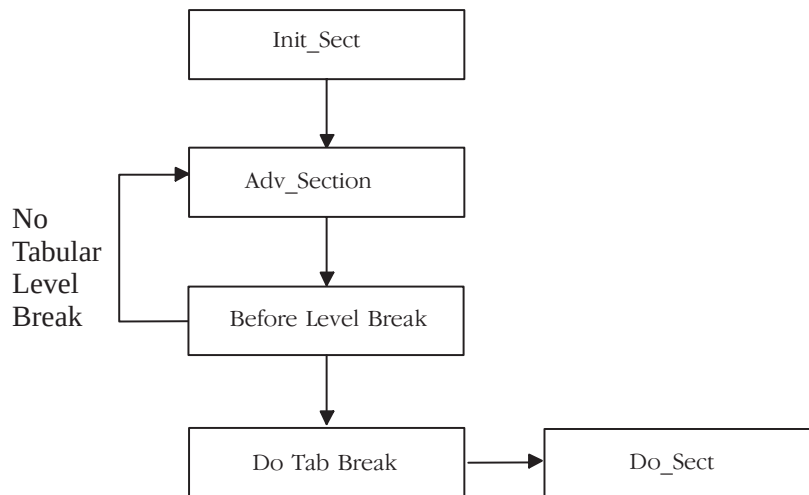
The following diagram illustrates the typical event flow for group and columnar sections.



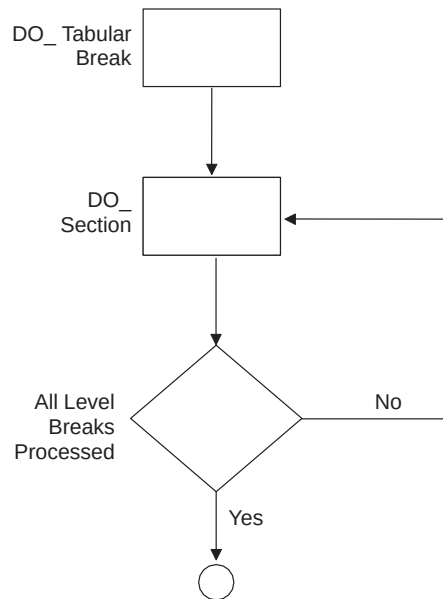
Processing Tabular Sections

Tabular sections are comprised of columns, rows and cells. To define what information is contained in each column, row, or cell, you define inclusion rules or calculations. An inclusion rule is either a set of criteria, a business function, or a named event rule for that row or column. One advantage of tabular sections is that you do not have to define additional sections for level-break logic or processing as would have to be defined with a group or columnar section. However, you must set up the data sequencing correctly because the level-break part of the section is dependent on it.

The following diagram illustrates the typical event flow for tabular sections.



As shown in the following graphic, tabular sections summarize information and print only when there is a level break. Because level breaks are dependent on data sequencing, your report can be as detailed as your data sequencing allows or as general as is needed. For example, suppose you sequence your data in a tabular section by Company only, but you have the objects Object Account and Subsidiary included in your report as well. If you sequence by Company only, and only two company records exist, the tabular section prints only those two records on the report. If you sequence by Company and Object Account, the tabular section prints both Company and Object Account information. Likewise, if you sequence on Company, Object Account, and Subsidiary, the system prints information for all three objects. Therefore, the more data you sequence and level break in your report, the more detailed your report will be.



Another advantage of using a tabular section is when you want to attach a level-break header or footer to a tabular section. You only need to define the level-break header or footer once and attach it to the lowest level-break object. When the lowest object level-breaks, the system processes the logic in the level-break section.

Other features in tabular sections include Drill Down, Account Level of Detail, a summarization feature, and automatic totaling. You can use Drill Down to provide a shortcut to an interactive application from the viewable output of a report. The interactive application shows the detail for the balances on the viewable output of your report. You can use the summarization feature (After Last Object Printed) to print or process logic after a section has been processed. Automatic Totaling automatically totals any numeric values, regardless of what type of numeric values they are. However, keep in mind that the system will try to total any value in your report (numeric or otherwise). To keep the system from totaling a particular column, open that column's properties and choose the Suppress Printing at Totals option.

You might have a situation in which you process a report using multiple sections or a combination of group and columnar sections. In this case, you might be able to use one tabular section in place of the multiple sections. An advantage of replacing multiple sections with a tabular section is that it improves your system's performance. Instead of calling multiple sections, your system calls only one section.

Attaching Logic to Joined Sections

If you join sections and want to attach logic to the joined sections, you must attach it to the Refresh Section event, not the INIT_Section event, to ensure that the logic is processed every time the section is processed. When joined sections are processed, the system initializes the INIT_Section event the first time the parent/child section is processed, then initializes the Refresh Section event for all subsequent times the parent/child section is processed. Therefore, if you attach an event to INIT_Section, the

logic is processed only once. If you attach it to the Refresh Section event, it is processed each time the section is processed.

See Also

The Enterprise Report Writing Guide

Data Selection and Sequencing

If your report or batch process takes a long time to process because it contains multiple sections, you might want to check the data selection for each section because the system will process all the records in the table, unless you specify otherwise. If you want a level-one section to adopt the data selection and sequencing from another section, you can use the Use Data Sel/Seq system function to propagate the data selection and sequencing to other sections. This functionality only affects the level one sections in your report.

For example, in the Pick Slips Print program (R42520), you can choose from the hundreds of columns in the F4211 table for data selection. In the first section, which is hidden and has database output to the F4211 table, the report must first process “commitments,” which could add rows to the F4211 table. The next section in the report needs to display the modified and updated F4211 table to show the committed records. Possible solutions are to use a temporary file or to change the data selection on multiple sections. In this case, you would use the Use Data Sel/Seq from a section’s system function, which adopts the data selection and sequencing from a previous level-one section.

The Use Data Sel/Seq system function allows a section to adopt the sequencing or data selection specifications from another section in the same report or from a section in an entirely different report. The target section can adopt the data selection criteria, the sequencing information, or both, from the source section. The selection or sequencing information from the source section replaces the information contained in the target section’s specifications.

To use the Use Data Sel/Seq system function, access the event rules for the Initialize Section event of the “target” section. The Initialize Section event is the only event that should invoke this system function.

How OneWorld Allocates Memory for Sections

OneWorld allocates memory for all sections and those allocations are destroyed at the end of reports.

See Also

- *Understanding Level-Break Processing* in this guide for more information about level-break headers and footers, and level-break events.
- The *Enterprise Report Writing Guide* for more information about data selection and sequencing; and for information about creating group, columnar, and tabular sections.

Understanding Custom Sections

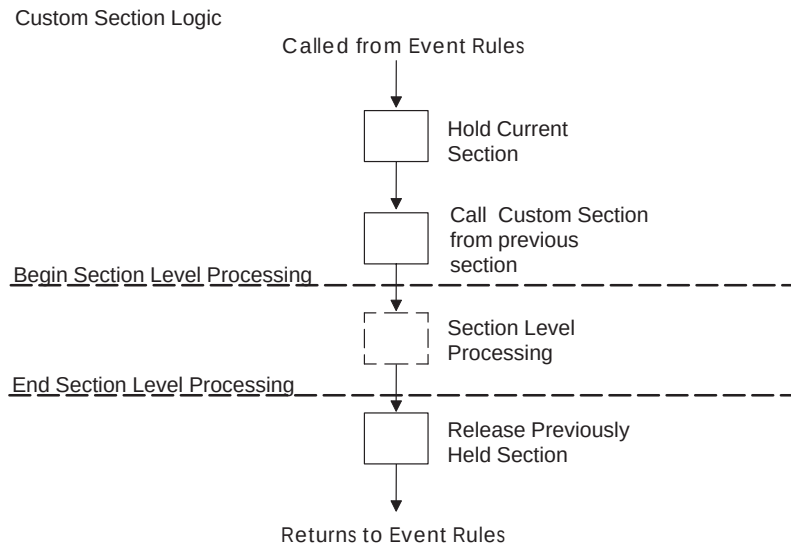
Custom sections allow you to control, through event rules, the information that prints on a report. You can use custom sections to force a page break by creating a custom section with no objects, then activate Page Break After Print in that section's Section Properties. You can also use custom sections to print variable text, or for sections that present the same information, but are formatted differently. For example, a report that exists in two different modules, but depending on a user's needs, calls a different section that displays information specific to that particular module.

Another example of using a custom section is when you want to print additional information depending on certain criteria. For example, you might have an accounts receivable report that shows a customer's payment history, but you want the batch engine to print additional information if a customer is delinquent in his payments. In this case, you can create a custom section to display if the customer is 30, 60, or 90 days overdue, then attach a processing option that calls that custom section. When the processing option is activated, and when the batch engine encounters a record that contains overdue information, the custom section prints; if no overdue information exists, the custom section does not print.

You can use the `DO_CUSTOM_SECTION` system function in columnar, group, or tabular sections to call the custom section. There is no limit to the number of custom sections you can use in a report. Custom sections are allocated and treated like any other level-two section.

Custom sections can contain business view fields, variables, or data dictionary fields. Your report processes the same way regardless of the objects contained within it.

The following graphic illustrates the process flow for custom sections. The batch engine stops processing the current section when it encounters a call for information contained within a custom section. After it processes the custom section, it returns to the previously held section.



The following topics are described:

- ☐ Attaching logic for a custom section
- ☐ Using a custom section

Attaching Logic for a Custom Section

Custom sections are launched using a system function in Event Rules. You attach the logic for a custom section to the section preceding the custom section. For example, if you want a custom section to process before a columnar section, you call the custom section using the system function `DO_CUSTOM_SECTION` from the columnar section's `DO_SECTION` event. Likewise, if you want a custom section to process before a level-break footer, you call the custom section using the system function `DO_CUSTOM_SECTION` from the `INIT_LEVEL_BRK_FOOTER` event.

You can call a custom section event from any section. Furthermore, you can use any event rule logic along with a custom section, such as If and While statements, business functions, table I/O, and so on.

When you run the report, the batch engine calls and processes each section until it encounters the system function for the custom section. The batch engine then retrieves the information for the custom section and processes it. When it finishes processing the custom section, it returns to the previous section unless instructed to do otherwise by another event rule.

You can call a custom section from any event rule except `INIT_Section`. If you try to call a custom section using `INIT_Section`, your report will not process.

The custom section process flow occurs in the same order as the type of section it is defined as, for example, group, columnar, or tabular.

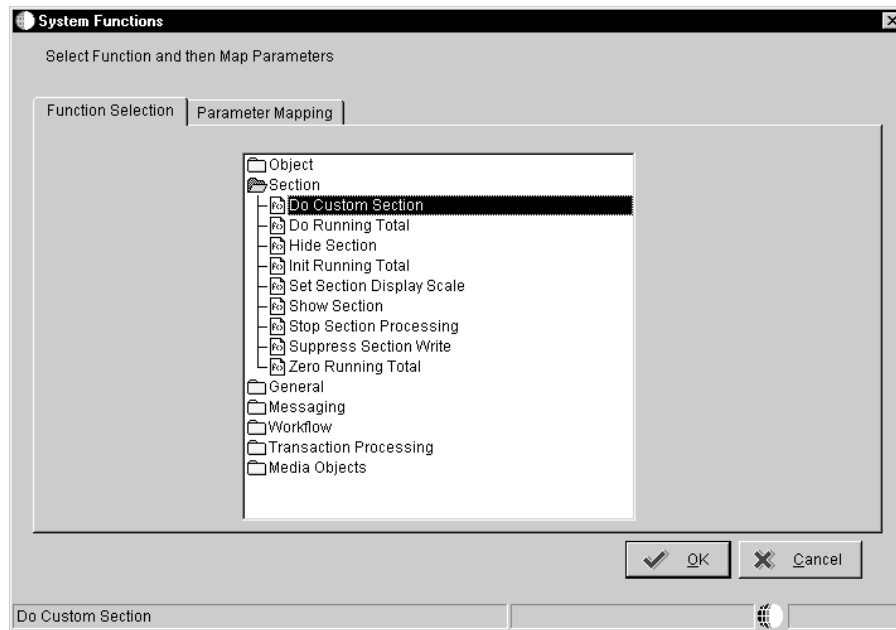
Using a Custom Section

When you use a custom section you specify the section as Conditional in Section Properties, then attach the system function event rule DO_CUSTOM_SECTION on the section preceding the custom section.

► To use a custom section

On Report Design

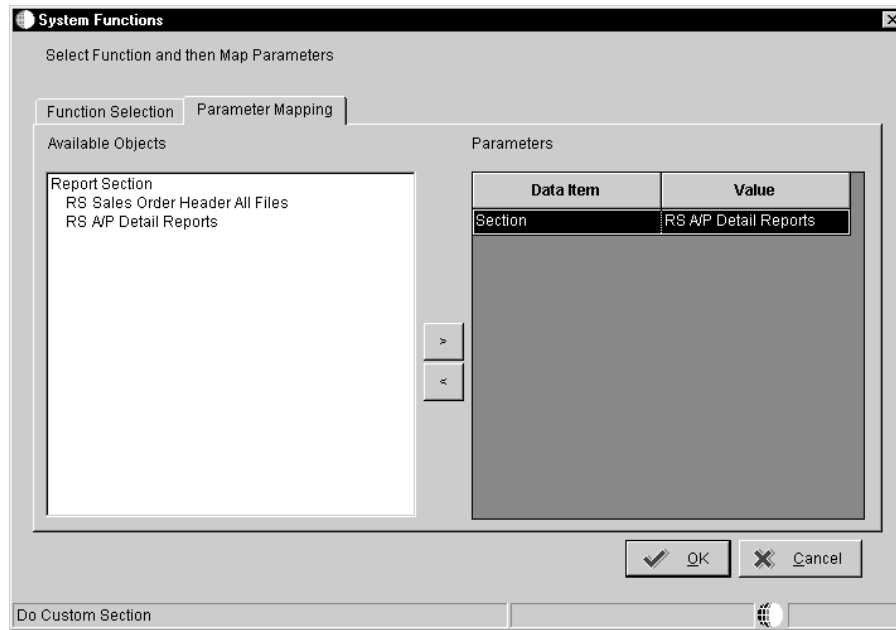
1. On the section from which you want to call the custom section, choose Event Rules from the Edit menu.
2. Choose the event at which you want to process the custom section.
3. Click the System Function button.
4. On System Functions, choose Section from the System Functions tree view, then choose Do Custom Section.



The system populates the Available Objects window with the sections in your report.

All sections in your report appear in the Available Objects window, not just the ones that you have specified as conditional.

5. Choose the conditionalized section you want to define as a custom section and click the right arrow to move it to the Parameters window.



6. Click OK.

Understanding Level-Break Processing

You use level breaks to identify a change in a report or batch process's data by comparing the previous record to the current record. If there is a change between the records, a level break occurs. You can use level breaks to segregate large report listings into smaller, logical groups, such as grouping telephone numbers by area code or employees by department. You can set up level breaks to initiate page breaks, totals, headers, or footers. Level breaks can only be attached to business view fields and not to any other variables in the report.

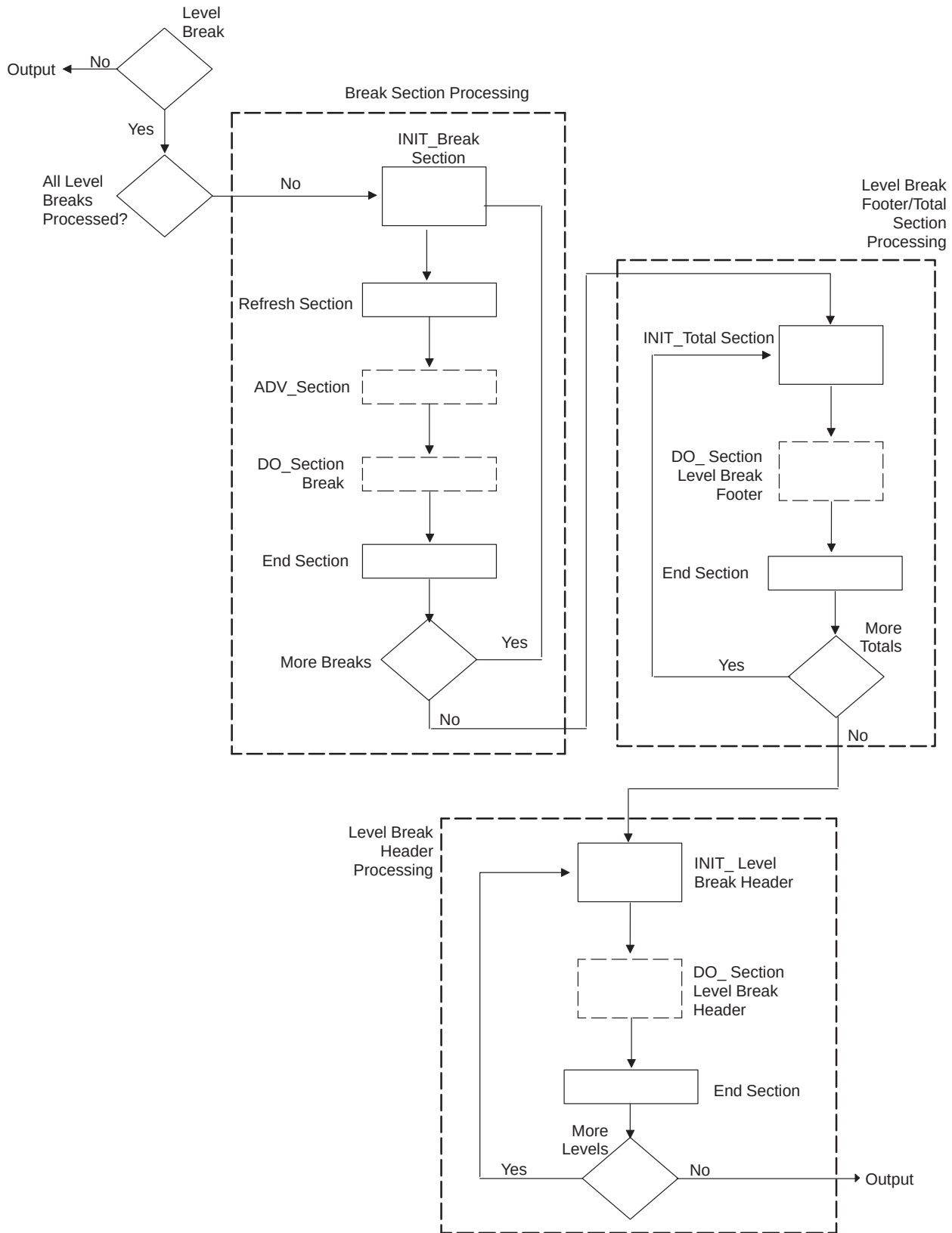
You can attach only one level-break header or level-break footer to one data item. If you try to attach more than one level-break header or footer to a data item, the batch system processes only the first level-break header or footer section that is attached to that item and ignores any other level-break sections that you have attached.

See Also

- *Creating Level-Break Headers and Footers* in the *Enterprise Report Writing Guide* for more information about level-break headers and footers.

The following graphic illustrates level-break logic processing. Before a level-break header or footer is processed, the batch system issues the INIT_Break Section event, which stops processing for the current section and moves to the level-break footer section. After the level-break footer section has been processed, the batch system processes the level-break header, if specified in the report.

Level-Break Logic Processing



Level-Break Header Sections

You use level-break header sections to present information at a level break within another section. For example, you can use a level-break header section in a telephone listing to announce the next group of phone numbers in a particular area code.

When you select the level break item, the level break header section assumes the business view for the section that the item resides in.

Using Level-Break Headers with Page Breaks

You can use a level-break header with a page break to force the level-break information to print on a new page. For example, if you place a level-break header and page break on the Customer data item, each time the batch system encounters a new customer record it will advance to the next page and print that information.

Level-Break Footer Sections

You use level-break footer sections to present information after a level-break has occurred in the preceding section. For example, you can use a level-break footer section in an employee listing to produce salary totals by department.

Conditionalizing Level-Break Headers or Footers

You can conditionalize a level-break header or level-break footer section so that it will print only if you tell the batch system to do so through a processing option. For example, you can create a report in which you want a total to print only when called by attaching the Hide Section system function to the level-break header or footer section in Event Rules. Then, in the Do_Section event you add an event rule for a processing option that will show or hide the section depending on the user's preference.

Level-Break Section Events

When you process a report that contains level-break headers and footers, OneWorld automatically calls the INIT_Level Break Footer Section or INIT_Level Break Header Section event. (If you are processing a tabular report, the batch system calls the INIT_Total Section event in place of the INIT_Level Break Footer Section event.) You can attach event rules to these events to control how the batch system processes them.

For example, if you want to create a report that summarizes information by company, you can design the report to read the detail file by company using a level-break footer. You set a level break on the Company data item, and when the company changes, the batch system calls the level-break footer for the detail section.

You can attach business functions, table I/Os, or other events to level-break section events.

INIT_Level Break Footer Section Event

When the batch system calls the INIT_Level Break Footer Section event, the values in the previous section are accessed and are either summarized or totaled. For example, if you have a level-break footer set up to total the revenue for a company, the level-break footer inserts the total below all the records that have been processed in the previous section. The INIT_Level Break Footer Section event is attached to the level-break footer section.

If you set up multiple level-break footers within a report, the batch system processes the footers starting from the lowest section up to the highest section.

If you need to attach logic to a level-break footer, attach it on the End Level Break Footer Section event rather than on the End Section event.

INIT_Level Break Header Section Event

The INIT_Level Break Header Section event is called after the previous section finishes processing. It is also called after the INIT_Level Break Footer Section event, if you have set one up. The INIT_Level Break Header Section event locates the header section associated with the current level and processes the information in the level-break header.

For example, if you have a level-break footer set up to create a grand total for a section, it will process before the level-break header for the next section, which might contain another company's revenues. The header section processes new records, or different data, from the previous section.

If you have set up multiple level-break headers within a report, the batch system processes the headers starting from the highest section down to the lowest section in the report.

END_Level-Break Header and Footer Section Events

After the Initialize Level Break events are processed, OneWorld launches the END_Level Break Header or Footer Section event, which ends the level-break process and returns to the previous section. You can attach logic that calls a custom section to the End Level Break Section event. For example, you might want to print generic text, a disclaimer, special text, and so on, after information in a level-break footer. In this case, you can create a custom section that contains the information you want to print, then attach the custom section logic to the End Level Break Section event. When you process the report, the disclaimer appears below the level-break footer section.

Understanding Batch Events

As a report or batch application is processed, there are certain points in the execution of the report or batch application at which the runtime engine pauses to process logic that has been attached. These points are called events, and you can use these events to insert custom logic for processing. OneWorld provides you with a set of predefined batch events. The basic flow of these events within a section is Initialize, Do, then End.

The engine processes some events only if the appropriate type of section exists. For example, INIT_Report Header and END_Report Header are initialized only if there is a report header section. Likewise, INIT_Level Break Footer and END_Level Break Footer are processed only if there is a level-break footer.

The Do_Section Event

The DO_Section event is invoked after the system has assigned new values to objects in the report and immediately prior to processing objects within a section. You most commonly attach logic to the DO_Section event because it occurs before any objects are processed, and most often the logic you attach at this point will affect the objects in some way.

The system processes the column headings first, then runs INIT_Object and DO_Object for each object in the section. When all objects have been fetched from the database, the engine runs End_Object.

If the object will not fit on the page, the engine invokes Suspend Object, which moves the object to the next page.

If the object is a child section, such as in a joined section, the system invokes INIT_Object the first time it is processed. For any subsequent times the child is processed, the engine invokes Refresh Section.

Additional Batch Events

In addition to the basic batch events you typically use, there are other events that give you flexibility in your batch processes.

Section Level Events

Advance Section	Occurs each time you do a fetch from the database. Use this if you need to perform processing on objects before a fetch. If this section does not have a business view attached, then this event is processed once.
After Last Object Printed	Occurs after a row is printed to an output file. Use this to process information after a row has been output.
Before Level Break	Use this to do processing after a fetch, but before any level breaks are checked.
Clear Space	Occurs when a section reaches its end. Use this to perform processing after a section is complete.
Do Section	Occurs after Advance Section after values have been assigned to print out to a printer or an output file. This event occurs before any information for the current record is written to the PDF file. This event occurs before Do Cell (if Tabular cells exist) and before Do Variable/Do Constant.
Do Tabular Break	Only valid for tabular sections. Occurs when any of the business view fields set as level breaks change. Use this to do processing that requires a change of values in any of the level break fields.
End Break Section	Occurs after a level break finishes. Use this to do processing immediately after a level break.
End Lvl Brk Footer Section	Use this to do processing immediately after a level break footer.
End Lvl Brk Header Section	Use this to do processing immediately after a level break header.
End Section	Occurs after a batch process has completed processing the last set of section values. Use this to do processing immediately after a section ends. This event is useful for last record and end of file procedures.
Init Break Section	Occurs after a level break begins processing. This event initializes a child section that is joined to the parent section on a level break.

Init Lvl Brk Footer Section	Use this to do processing immediately before a level break footer.
Init Lvl Brk Header Section	Use this to do processing immediately before a level break header.
Initialize Section	Occurs when a batch process encounters a section for the first time. Use this to do processing immediately before a section begins. This is useful for working with global variables or performing other preparatory procedures. For conditional sections, this event will be processed each time the section is called.
Refresh Section	The first time the UBE encounters a child section, it issues an initialize section event. Each subsequent time the child section is to be processed, the batch process uses Refresh Section. At this point, the internal structures and pointers for the child section have been established and the UBE is about to select a new group of records for the child section. This logic also works for the level break sections. Use this to set the object values of level two sections based on the parent section. You can also use this event to reset or modify data selection and sequencing of the child section.
Suspend Section	Processes when an overflow page break occurs. For example, if information being printed does not fit in the space available on a page. This temporarily stops the section processing. This temporarily stops the section processing. Use this do do processing when a page break occurs.

Page Header Section Level Events

Initialize Page Header	Occurs at the beginnning of a report after any report header logic and before the page header section processes for the first time. It also processes every time a page break occurs. Use this to initialize values that cannot be set until after the report header logic executes. This is similar to Init Section for a normal group, columnar, or tabular section, except that it is only processed for a page header section.
End Page Header	Occurs after the page header finishes processing. Use this to do processing immediately after a page header.

Page Footer Section Level Events

Initialize Page Footer	Occurs at the beginning of the report after any report header logic and before the page header section processes for the first time. Use this to initialize values to be printed in the current page footer section. These assignments typically depend on information processed so far on that page.
End Page Footer	Occurs after the page header finishes processing. Use this to do processing immediately after a page header.

Report Header Section Level Events

Initialize Report Header	Processes once at the very beginning of the report before anything else in the report processes. Use this to initialize values at the beginning of a report. This is similar to Init Section for a normal group, columnar, or tabular section except that it only processes for a report header section.
End Report Header	Occurs after the report header processes. Then the report processes the page header for a report. Use this to do processing immediately after a report header.

Report Footer Section Level Events

Initialize Report Footer	Occurs once at the very end of a report after everything else processes and before the report footer prints. Use this to initialize values to print in the report footer.
End Report Footer	Occurs after the report footer processes. After processing finishes, the report terminates. Use this to do processing immediately after a report footer.

Variable Events

Do Variable/ Do Constant	Occurs just before the font and color are selected and before the value of the object is translated into a printable string of characters and output to the page. Use this to do processing after an object has been processed. This is your last change to manipulate the value or display attributes of the objects before output.
End Variable/ End Constant	Occurs immediately after an object processes even if the object is invisible or suppressed. Use this to do processing after an object processes.
Initialize Variable/ Initialize Constant	Occurs each time a report object or variable is to be processed. Use this to do processing before an object processes. This is useful to do processing that affects the calculation of an object's position because the object's position on the page has not yet been determined.
Skip Variable/ Skip Constant	Occurs when an object will not fit on the current page. The batch process issues a skip variable to bypass the object until the next page begins processing. Use this to change the value of an object at the page break.
Suspend Variable	If an object requires multiple text strings or column headings and if only part of the object fits on a page, then the batch process issues a suspend variable to halt processing of the object until the next page has been started. Use this to modify the value at the page break. Because the value of the object has already been partially processed, this is not a good time to manipulate that value.
Column Inclusion	Only valid for tabular sections, this event occurs after a record is fetched from the database. Use this event to perform calculations on financial data.
Do Cell (Tabular)	Occurs during Do Object after processing calculations for a cell. Use this event to manipulate cell data before displaying it. This event occurs before Do Variable/Do Constant. This event occurs during calculations.

Variable Population at Specific Events

The following table details the event names that are associated with each type of object within the Report Design Aid. Next to each event, the table indicates which kinds of

variables are available for use within event rules for that combination of event and object.

BC Business View Column.

Global RV Global Report Variable. Global Report Variables are available to the entire report.

Local RV Local Report Variable. Local Report Variables are only available for the section they reside in.

Section/Section Group	Events in Sequence	BC	Global RV	Local RV
Report Header	1. Initialize Report Header	No	Yes	No
	2. Do Section	No	Yes	Yes
	3. After Last Object Printed	No	Yes	Yes
	4. End Report Header	No	Yes	Yes
Page Header	1. Initialize Page Header	No	Yes	No
	2. Do Section	No	Yes	Yes
	3. After Last Object Printed	No	Yes	Yes
	4. End Page Header	No	Yes	Yes
Columnar Section and Group Section	1. Initialize Section	No	Yes	No
	2. Advance Section	Yes	Yes	Yes
	3. Before Level Break	Yes	Yes	Yes
	4. Do Section	Yes	Yes	Yes
	5. After Last Object Printed	Yes	Yes	Yes
	6. End Section	Yes	Yes	Yes
	Suspend Section - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Refresh Section	Yes	Yes	Yes
	Clear Space	No	Yes	No
	Init Break Section	No	Yes	No
	End Break Section	No	Yes	No
	Not Used:			
	Init Lvl Brk Footer Section			
	End Lvl Brk Footer Section			
	Init Lvl Brk Header Section			
	End Lvl Brk Header Section			

Section/Section Group	Events in Sequence	BC	Global RV	Local RV
Tabular Section	1. Initialize Section	No	Yes	No
	2. Advance Section	Yes	Yes	Yes
	3. Before Level Break	Yes	Yes	Yes
	4. Do Tabular Break	Yes	Yes	Yes
	5. Do Section	Yes	Yes	Yes
	6. After Last Object Printed	Yes	Yes	Yes
	7. End Section	Yes	Yes	Yes
	Suspend Section - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Refresh Section	Yes	Yes	Yes
	Clear Space	No	Yes	No
Variables	Init Break Section	No	Yes	No
	End Break Section	No	Yes	No
	Do Balance Auditor	Yes	Yes	Yes
	1. Initialize Variable	Yes	Yes	Yes
	2. Do Variable	Yes	Yes	Yes
Column Headings	3. End Variable	Yes	Yes	Yes
	Suspend Variable - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Skip Variable	Yes	Yes	Yes
	Column Inclusion	Yes	Yes	Yes
	Do Cell	Yes	Yes	Yes
	1. Initialize Constant	Yes	Yes	Yes
	2. Do Constant	Yes	Yes	Yes
	3. End Constant	Yes	Yes	Yes
	Suspend Constant - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Skip Constant	Yes	Yes	Yes
	Do Column Heading	Yes	Yes	Yes

Section/Section Group	Events in Sequence	BC	Global RV	Local RV
Level Break Header	1. Initialize Section	No	Yes	No
	2. Do Section	Yes	Yes	Yes
	3. After Last Object Printed	Yes	Yes	Yes
	4. Advance Section	Yes	Yes	Yes
	5. End Lvl Brk Header Section	Yes	Yes	Yes
	Suspend Section - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Refresh Section	Yes	Yes	Yes
	Clear Space	No	No	No
	Init Break Section			
	End Break Section			
	(never hit in a L.B.Header): Init Lvl Brk Header Section Before Level Break Init Lvl Brk Footer Section End Lvl Brk Footer Section End Section			
Level Break Footer	1. Initialize Section	No	Yes	No
	2. Do Section	Yes	Yes	Yes
	3. After Last Object Printed	Yes	Yes	Yes
	4. Advance Section	Yes	Yes	Yes
	5. End Lvl Brk Footer Section	Yes	Yes	Yes
	Suspend Section - Occurs when object won't fit on the current page	Yes	Yes	Yes
	Refresh Section	Yes	Yes	Yes
	Clear Space	No	Yes	No
	Init Break Section			
	End Break Section			
	(never in a L.B.Footer): Init Lvl Brk Footer Section Before Level Break Init Lvl Brk Header Section End Lvl Brk Header Section End Section			
Page Footer	1. Initialize Page Footer	No	Yes	No
	2. Do Section	No	Yes	Yes
	3. After Last Object Printed	No	Yes	Yes
	4. End Page Footer (not in RDA)	No	Yes	
Report Footer	1. Initialize Report Footer	No	Yes	No
	2. Do Section	No	Yes	Yes
	3. After Last Object Printed	No	Yes	Yes
	4. End Report Footer	No	Yes	Yes

System Functions within Batch Events

Batch system functions give you flexibility and control over how your reports are processed. For example, you can use batch system functions to hide or show an object, to hide or show a section, to generate a message, and so on.

The batch system functions you might use consist of the following main categories:

Object	You can use Object system functions to do things such as hide or show objects.
Section	You can use Section system functions to do things such as hide or show sections or work with totals.
General	You can use general system functions to do things such as work with selection and sequencing.
Messaging	You can use message system functions to do things such as send, update, or delete messages.
Workflow	You can use workflow messages to do things such as work with processes.
Transaction Processing	You can use transaction processing system functions to begin, commit, or rollback transactions.
Media Objects	You can use media object system functions to work with media objects.

Some general batch system functions that are commonly use are:

Set Sequence Append Flag	This system function allows you to append or add the sequence for a section to the report's sequencing.

Stop Section Processing

Stops processing the current section and goes to the next section. This system function is helpful for performance, especially when there is a large amount of event rule logic that remains to be performed. For example, if no more customers exist with a credit limit over a certain amount, the system stops processing that section and moves to the next section.

Stop Section Processing differs from Suppress Section Write in that Suppress Section Write only suppresses the current record, which causes the engine to process the same section for the next record.

Hide Object

For group and tabular sections, when you hide objects using the Hide Object system function in Event Rules, be aware that even though it might be the only object on that line, the system still prints a blank line for it. The system has no way of knowing whether there are any other objects on that line that need to be printed. To keep the system from printing a blank line, place the object in its own conditional section and suppress the printing of that section using the Hide Object system function.

Other system functions are published online under *Published APIs*.

Understanding Batch Runtime Processing

Batch runtime processing refers to how events, such as initializing a section, and their attached event rule logic are evaluated at runtime.

Runtime structures are structures or blocks of memory that hold data in memory as the data is read, processed and written to the database.

Report Design provides several different field types and event rules that are associated with them.

Available Objects

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 a report or batch process at runtime.

During runtime processing, data is stored in memory in an internal data structure. Certain fields of the data structure temporarily store data during runtime until it is no longer needed. Then data can be cleared to process another record.

The following available objects are defined for batch processing:

BC	A column in the business view. Business view columns will appear in this list. These columns are filled with values from the database when a fetch is performed and are the values saved to the database on an add or update.
PO	A value passed from a processing option. These values are passed into the application when it is started and can be accessed by any form in that application. They could have been entered by the user, or they could have been set up in a particular version of an application.
VA	Event rules variables. These objects represent any variables set up by the developer in event rules. They are not manipulated by the system.
SV	System variables. These objects represent some environment variables that have been made accessible to event rules.

SL	System literals. These objects represent some constant system values that have been made accessible to event rules.
TV	Text Variable
RC	Report Constant (UBE)
RV	Report Variable (UBE)

Processing Available Batch Objects

When the value of an available object is changed during event rule processing the new value is copied to the business view data, section data, or processing option data *immediately after that ER line is processed* (internal runtime structure). Whenever the RC runtime data is changed, any reference to BC will reflect the same value. This change in the RC is not reflected in the report until the RC runtime value is moved to the screen.

Typical Event Flow for a Group Section

The runtime engine processes events in a certain order. The typical events for a group section and the order in which they are processed follow.

The following example represents how values in the runtime structures are stored in memory compared to how they appear in the report. This example uses a group section for an Address Book report.

Initialize Section

The following graphic illustrates what values are in the runtime structures after the following events occur.

- Initialize Section

RC		BC	
AN8	Address Number	AN8	
ALPH	Alpha Name	ALPH	

Advance Section

The following graphic illustrates what values are in the runtime structures after the following events occur.

- Initialize Section
- Advance Section

RC		BC	
AN8	Address Number	AN8	
ALPH	Alpha Name	ALPH	

Before Level Break

The following graphic illustrates what values are in the runtime structures after the following events occur.

- Initialize Section
- Advance Section
- Before Level Break

RC		BC	
AN8	Address Number	AN8	1
ALPH	Alpha Name	ALPH	Financial/Distribution Company

Do Section

The following graphic illustrates what values are in the runtime structures after the following events occur.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section

RC		BC	
AN8	Address Number	AN8	1
ALPH	Alpha Name	ALPH	Financial/Distributino Company

After Last Object

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object

RC		BC	
AN8	Address Number	AN8	1
ALPH	Alpha Name	ALPH	Financial/Distribution Company

Report	
Address Number	1
Alpha Name	Financial/Distribution Company

Advance Section

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section

- After Last Object
- Advance Section

RC		BC	
AN8	Address Number	AN8	1
ALPH	Alpha Name	ALPH	Financial/Distribution Company

Report	
Address Number	1
Alpha Name	Financial/Distribution Company

Before Level Break

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- Before Level Break

RC		BC	
AN8	Address Number	AN8	27
ALPH	Alpha Name	ALPH	Eastern Area Distribution Cent

Report	
Address Number	1
Alpha Name	Financial/Distribution Company

Do Section

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- Before Level Break
- Do Section

RC		BC	
AN8	Address Number	AN8	27
ALPH	Alpha Name	ALPH	Eastern Area Distribution Cent

Report	
Address Number	1
Alpha Name	Financial/Distribution Company

After Last Object

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- Before Level Break
- Do Section
- After Last Object

RC		BC	
AN8	Address Number	AN8	27
ALPH	Alpha Name	ALPH	Eastern Area Distribution Cent

Report	
Address Number	1
Alpha Name	Financial/Distribution Company
Address Number	27
Alpha Name	Eastern Area Distribution Center

Advance Section

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section

RC		BC	
AN8	Address Number	AN8	27
ALPH	Alpha Name	ALPH	Eastern Area Distribution Cent

Report	
Address Number	1
Alpha Name	Financial/Distribution Company
Address Number	27
Alpha Name	Eastern Area Distribution Center

End Section

The following graphic illustrates what values are in the runtime structures after the following events occur and what the report will look like at this point.

- Initialize Section
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- Before Level Break
- Do Section
- After Last Object
- Advance Section
- End Section

RC

AN8	Address Number
ALPH	Alpha Name

BC

AN8	27
ALPH	Eastern Area Distribution Cent

Report

Address Number	1
Alpha Name	Financial/Distribution Company
Address Number	27
Alpha Name	Eastern Area Distribution Center



Database Output

You can use database output to update the database in conjunction with your report generation. You can attach a database output specification to any report section with a business view, but not a report or page header or footer.

You can use database output in Report Design to update, insert, or delete records in a table or text file. You can also use a special operation called “Insert or Update”. This operation attempts to insert a record into a table. If a record with the same primary key exists, the insert fails and the existing record is updated.

You can use database output to output data to OneWorld tables or to text files. Text files can be specified as either comma delimited or fixed length text records. The comma delimited format is useful for transferring data to spreadsheets.

OneWorld provides several tools you can use for batch database maintenance.

Table Conversion	Use for high-performance SQL table-to-table processing. Allows access to non-OneWorld tables. No reporting.
Database Output	Reporting and output must occur simultaneously. Source and destination tables may reside in different environments. Allows output to text files.
Table I/O (in Event Rules)	Reporting and output must occur simultaneously. Input data must be transformed before it can be output.

Database output is performed for every row of data processed in the section. All database operations occur on a record-by-record basis using the standard JDE Base middleware APIs.

This section describes the following topics:

- ☐ Defining database output
- ☐ Overriding Environments for database output



Defining Database Output

Defining database output involves defining the output and then overriding the specific environment.

Complete the following:

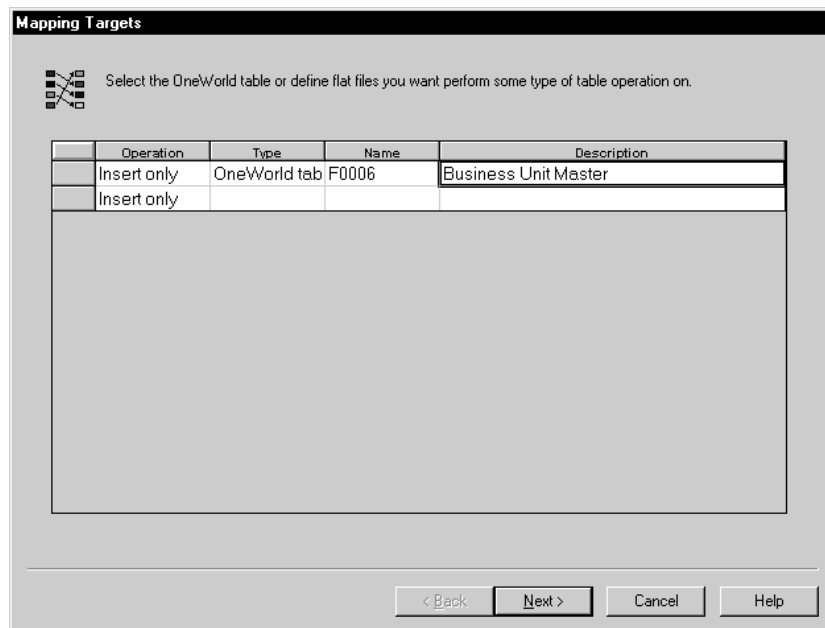
- Define database output
- Override environments

► To define database output

From a report section in Report Design

1. From the Section menu, choose Database Output.

The Mapping Targets form appears.



The Mapping Targets dialog box is shown. It has a title bar 'Mapping Targets'. Below the title bar, there is a small icon of a table and a text label: 'Select the OneWorld table or define flat files you want perform some type of table operation on.' Below this is a table with four columns: 'Operation', 'Type', 'Name', and 'Description'. The table contains two rows. The first row has 'Insert only' in the Operation column, 'OneWorld tab F0006' in the Type column, and 'Business Unit Master' in the Description column. The second row has 'Insert only' in the Operation column and is empty in the other columns. Below the table is a large empty rectangular area. At the bottom of the dialog box are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Operation	Type	Name	Description
Insert only	OneWorld tab F0006		Business Unit Master
Insert only			

2. Complete the following fields to define a list of output tables or files (targets).
 - Operation
 - Type

- Name

Operation	For a OneWorld table operations may be Insert, Update, Delete, or Insert or Update. For text files this must be insert.
Type	OneWorld table, comma delimited text file, or fixed record length text file
Name	Name of a valid OneWorld table or a valid file used for text file output

You can double-click on these fields to see a list of available choices. Double-click on your choice to select it. You can also type in the first few letters of your choice to see a list of available choices that start with those letters.

You can use the source target several times with different operations.

The Mappings form appears.

3. Click Next.

Target Column	Source Section	Source Object	Dictionary override
Business Unit			☐
Type Business Unit			☐
Description - Compres:			☐
Level of Detail			☐
Company			☐
Address Number			☐
Owner/Receivable Ad			☐
County			☐
State			☐
Model Account/Consol			☐

4. For each target, specify which of the available section columns or variables should be assigned (mapped) to which column in the target.

Double-click on a Source Section or a Source Object cell to display a list of options.

You can also turn data dictionary overrides on and off here.

Mappings



Edit assignments of source objects to target columns/fields.

Select the target table or file.

F0006

Double-click in the Source Section and Object columns to select the object whose value will be assigned to the Target Column.

Map Same

Target Column	Source Section	Source Object	Dictionary override
Business Unit	Job Cost Master		☐
Type Business Unit	Job Cost Master	[BC] Business Unit	☒
Description - Compres:			☐
Level of Detail			☐
Company			☐
Address Number			☐
Owner/Receivable Ad			☐
County			☐
State			☐
Model Account/Consol			☐

< Back

Finish

Cancel

Help

5. Click Map Same if you wish to map all columns.

Mappings



Edit assignments of source objects to target columns/fields.

Select the target table or file.

F0006

Double-click in the Source Section and Object columns to select the object whose value will be assigned to the Target Column.

Map Same

Target Column	Source Section	Source Object	Dictionary override
Business Unit	Job Cost Master	[BC] Business Unit	☐
Type Business Unit	Job Cost Master	[BC] Business Unit Typ	☐
Description - Compres:		[BC] Description - Com	☐
Level of Detail	Job Cost Master	[BC] Level of Detail - Bi	☐
Company	Job Cost Master	[BC] Company	☐
Address Number	Job Cost Master	[BC] Address Number	☐
Owner/Receivable Ad	Job Cost Master	[BC] Address Number	☐
County			☐
State	Job Cost Master	[BC] State	☐
Model Account/Consol	Job Cost Master	[BC] Model Accounts a	☐

< Back

Finish

Cancel

Help

6. Click Finish when you have finished your assignments.

Field	Explanation
Database File	The target is a pre-existing database table.

Field	Explanation
Flat File	The target is a flat file that will dynamically be created by the database output process.
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Insert Only	Inserts a new record into the target.
Update Only	Updates an existing record in the target.
Insert Or Update	Updates a record if it exists in the target or inserts a new record if it does not.
Map	Use this grid to assign available objects to output columns for the currently selected target. You can double click in the Source Selection or Source Object to display a list of available choices. Click in the Overrides column to display the Dictionary Overrides dialog.

See Also

- *Overriding Data Dictionary Triggers at Design Time* for information on how to override and disable data dictionary triggers.

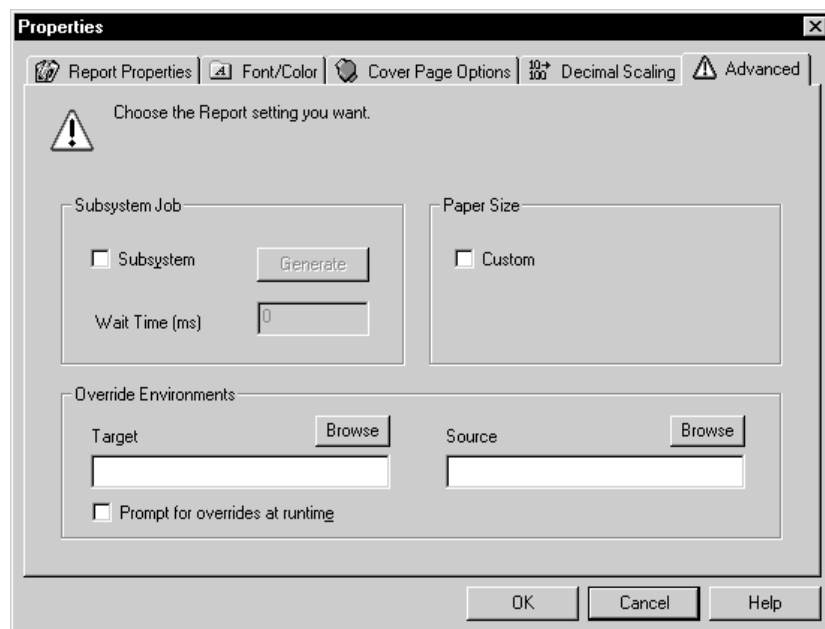
Overriding Environments for Database Output

You can override the environment you use for your database output.

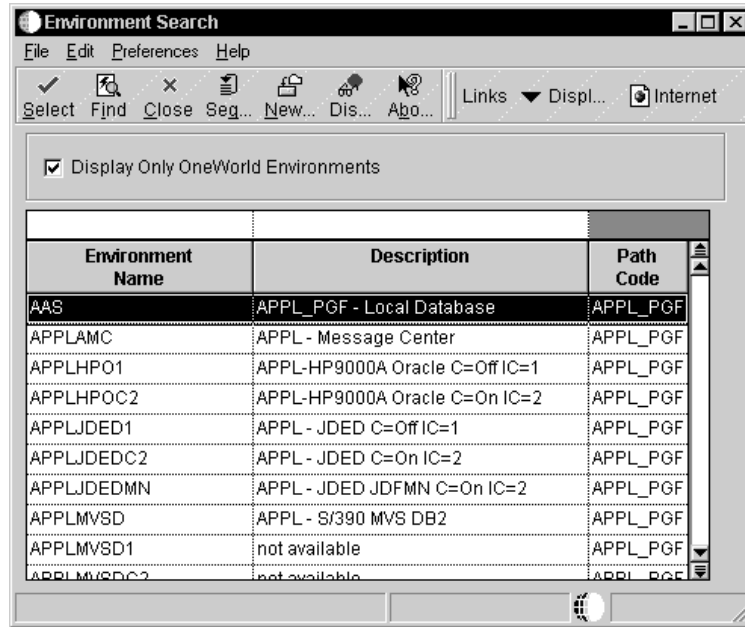
► To override environments for database output

From a report in Report Design

1. From the Report menu, choose Report Properties.
2. Click on Advanced.



3. On Report Properties, complete the following field or click the Browse button to find the source environment.
 - Source



4. Choose a source environment and click Select.
5. On Report Properties, click Prompt for overrides at runtime, if desired.
6. Choose a target environment and click Select.
7. To suppress the database output functionality, you can choose between two options:
 - If you do not want database output for the entire report, then override the database output in the version and delete the mapping record.
 - If you do not want database output for specific rows of data, then use the Suppress Section Write system function. This system function suppresses not only printed output but also database mapping.

Field	Explanation
Override Data Source	Specifies the name of the data source the report will use to access all tables. This will override the default data source for any table in the report.
Prompt At Run Time	Prompts the user at the time of report execution to specify the name of a data source or environment.



Subsystem Jobs

Within OneWorld, subsystem jobs are batch processes that continually run independently of, but asynchronously with, OneWorld applications. These OneWorld subsystem jobs function with the system's logical process or queue defined for the server platform. You can configure OneWorld to use one or more subsystems.

You use subsystem jobs to offload processor resources or to protect server processes. Examples of applications that are suited for use by subsystems include Distribution Warehousing, Inventory, and Sales Order Processing. For example, you might execute the Sales Order Entry application on a workstation and want to automatically print pick slips when all orders are entered. If you are using a version of pick slips that has the subsystem job function enabled in Report Properties, the request is executed by a OneWorld subsystem job. The pick slip request is routed to and processed by the subsystem job on the defined enterprise server. As a result, no additional processing resources are required of the workstation.

When a OneWorld application issues a request for a job to run in a OneWorld subsystem, it places a record in the subsystem table F986113. This record is identified by a subsystem job name and version, and contains status and operational indicators. Embedded in the record is key information that allows the OneWorld subsystem to process the record without additional interaction with the requesting application. The continuously running OneWorld subsystem monitors the records in this table. If the subsystem finds a record with its name, version, and appropriate status indicators, it processes the record and updates the status accordingly.

The following tasks are described:

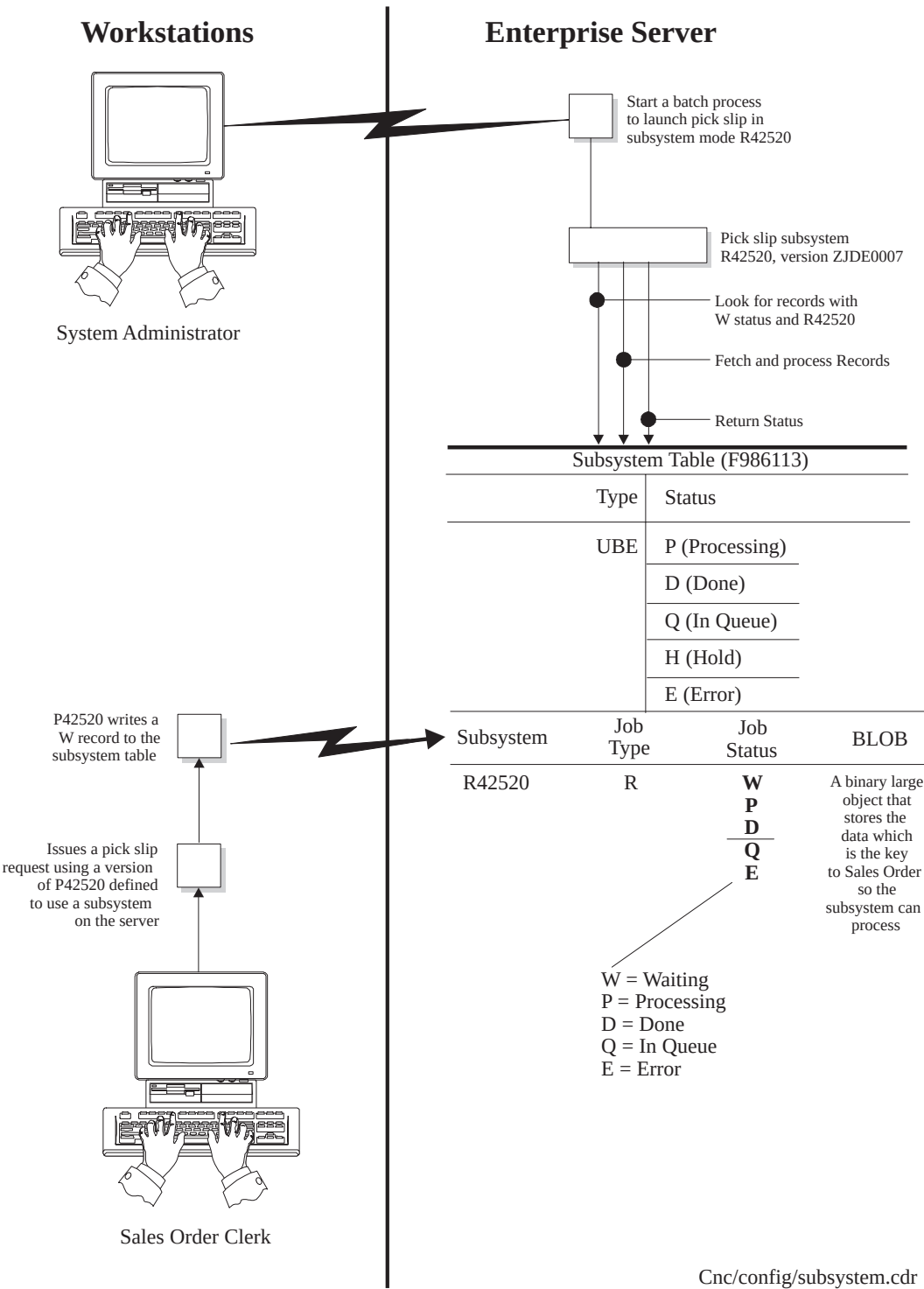
- ☐ Defining subsystem jobs
- ☐ Adding an API record to the subsystem table

See Also

- The *System Administration Guide* for more information about subsystem jobs.

The following graphic illustrates how the system processes a subsystem job.





Defining Subsystem Jobs

Subsystem jobs are continuous jobs that process records from a data queue. This type of job runs until you request an end to the job. Subsystem jobs read records one at a time for a subsystem table, retrieve information from the particular record, and run a configurable processing engine for each record. At the end of the records, instead of ending the job, subsystem jobs wait for a specific period and then retrieve the information for each record once again. For each subsystem job, there can be multiple records in the subsystem table.

You start a subsystem job as you would a regular batch job. There is no difference between running a subsystem job and running a batch job.

Before processing, OneWorld makes sure that limits for the subsystem job on the particular server have not been exceeded. If exceeded, the configurable processing engine will not process the subsystem job.

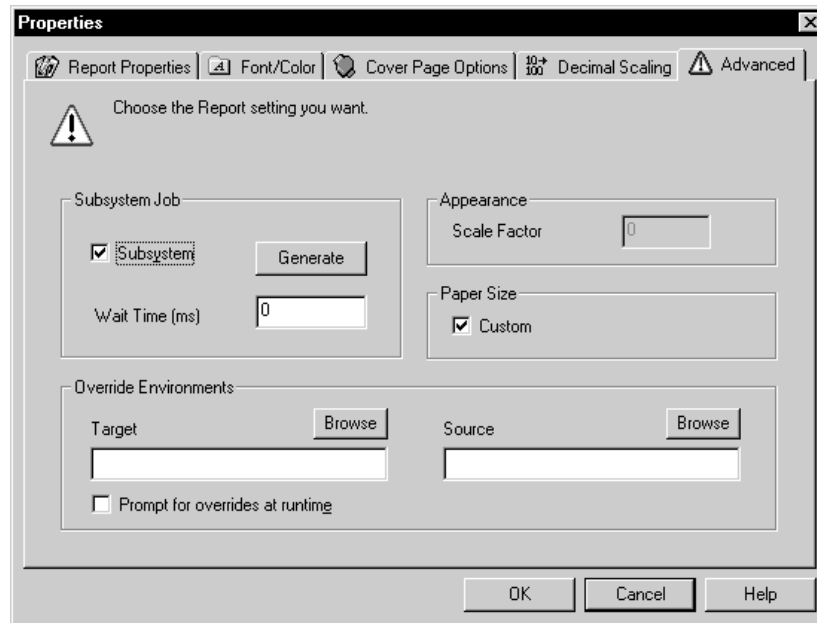


To define subsystem jobs

On Report Design with a report open

1. From the Report menu, choose Report Properties.

The Properties form appears.



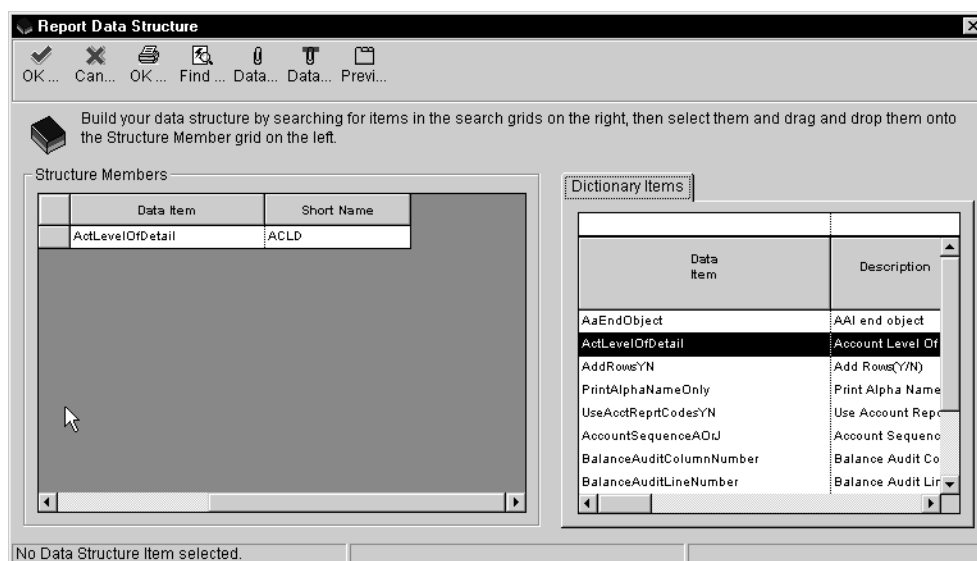
2. On the Properties - Advanced tab, choose the following options, then click OK:

- Subsystem
- Wait Time (ms)

The value in this field represents the time the subsystem report will check the subsystem file for new records to process.

3. On Report Design, from the Report menu, choose Report Data Structure.

The Report Data Structure form appears.



4. On Report Data Structure, click the Find button to display a list of data items.
5. From Dictionary Items, drag items into Structure Members, then click OK.
6. On Report Design, from the Report menu, choose Report Properties.
7. On the Properties - Advanced tab, click the Generate button to create a header file.

Note: The header file takes the form of *Report_Name.h* under the `\B7\$(environ)\include` subdirectory. In an interactive application, from which you will send a record to process in the subsystem job, you can use this header file inside a business function to create a subsystem record.

For example, by clicking the OK button on Sales Order Entry, you can specify to call a business function that uses this report's data structure along with the subsystem's application programming interfaces (APIs) to trigger the subsystem job.

Adding an API Record to the Subsystem Table

You add an API record to the subsystem table so that the subsystem job can perform the batch process. When you add a record to the subsystem table, you create a business function, then attach the header file that you generated to that business function using Microsoft Visual C++. After you attach the header file to the business function, you attach the business function to the event rule process of the program you want to call.

This API record retrieves data structure and user information from the cache. If the server name is not passed, the API finds Object Map Record from Object Configuration Table (F98611). If the record exists, it will use the record to send a JDENet Message to the server to add the record to the subsystem table on the server. However, if the user provides an Override Server name, the JDENet message is sent to that server instead.

Before You Begin

You must generate the header file using Report Data Structure to ensure that the API is called from the business function. See *Defining Subsystem Jobs*.

Example: Subsystem Job Header File

The following example shows what the subsystem job header file looks like.

```
#include <jde.h>
/*****
 * Report      : R98SSUBE
 * ReportId    : 8123244
 * DSTRId      : 380813
 *
 * Note:
 * Do not edit the following typedef
 * To make modifications, use the Report Design Aid Tool to
 * Generate a revised version.
 *****/
#ifndef REPORT_DS_380813
#define REPORT_DS_380813
typedef struct tagDS_RI_380813
{
    char                                ProgramId[11];
} DSRI380813, *LPDSRI380813;
#define                                IDERRProgramId_1
                                1L
#endif /* #define REPORT_DS_380813*/
#endif /* #define __R98SSUBE_H */
```

► **To add a record to the subsystem table**

1. In OneWorld, create a new business function.
2. In Microsoft Visual C++, open the business function include file and add the name of the header file you generated.

In the following example, lines that you enter are underlined.

```
/* *****  
 *  
/* Table Header Inclusions  
*****  
/  
  
/* *****  
*****/  
External Business Function Header Inclusions  
*****  
*****/  
#include <R98SSUBE.h>  
  
/* *****  
*****  
 * Global Definitions  
*****  
*****/  
  
/* *****  
*****  
 * Structure Definitions  
*****  
*****/  
 * TYPEDEF for Data Structure  
 * Template Name:  
Report Interconnect Data Structure  
 * Template ID:  
D983059  
 * Generated:   Wed Oct 18 14:01:22 1995
```

3. Open the business function source file in Visual C++ and declare the variable of this data structure type, then populate the members of the data structure.

```
#include <jde.h>

/*****
*****
* Variable declarations
*****
*****/

HUSER hUser=NULL;
LPSTR szServer=NULL;
DSRI380813 dsRI; /* Declare the variable of type REPORT
INTERCONNECT DATA STRUCTURE */
BOOL bRet=FALSE;
JDEDB_RESULT rcode;

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

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

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

if ((lpBhvrCom == NULL) ||
    (lpVoid == NULL) ||
    (lpDS == NULL))
```

4. In the source file, call the API to add the record.

```
*****
Main Processing
*****/
memset(&dsRI, 0, sizeof(DSRI380813));
/* Populate the members of the Report Interconnect Data
Structure */
strcpy(dsRI.ProgramId,lpDS->szString01);
/* Call Subsystem API to add the record to the Subsystem
Table */
/* Note : As Environment Name is set to NULL, this API will
use OCMto find the default Environment of this UBE */
bRet=jdeAddSubsystemRecord( hUser,/* User Handle */
    "R98SSUBE",/* Name of the subsystem */
    "XJDE0001",/* Name of the Subsystem Version*/
    NULL,/* Name of the override env - not
used */
    szServer,/* Name of the server */
    &dsRI);/* Subsystem Connect DS */
/*****
```

5. After you call the API to attach the record, attach the business function to the event rule process of the program you want to call. This program can be either an interactive or batch application.

See Also

- *Working with Business Functions*
- *Working with Event Rules Design*

Messaging



Messaging

Use J.D. Edwards messaging facilities to get pertinent information to the end user in the most effective and user-friendly manner. When you design an application to use messaging, you must evaluate what information is necessary for a user to accomplish a task. You can deliver a message in real time, where the message is displayed on the status bar. The method you use to provide information to the user depends on the situation:

- Use an interactive error message if there is an error during the entry of a record.
- Use an informational message sent to the Workflow Center if information needs to be conveyed and action requested.
- Use an alert message if information is urgent and needs immediate attention.
- Use a batch error message if errors are detected while a report is running.

There are three parts to creating system-generated messages:

- The message itself.

Do you require a simple message or a text substitution message? Are all the text substitution pieces available?

- The logic.

Has certain criteria been met so that a message should be sent? This will usually be event rule logic.

- Is this an action message?

Does the message require action by the users? Are all the required parameters available at the time the message is to be sent?

Message Types

There are two main types of messages:

- 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 be sure 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. Substitution values are inserted into the message for the appropriate variable at runtime. This gives the user a customized message unique to every instance of the message.

There are two types of text substitution messages:

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

They are both created in the same manner.

Error Messages

Error messages are stored in the data dictionary. The data dictionary design tools can be found on menu GH951.

Workflow Messages

When designing an application to use messaging, you must evaluate what information needs to be used to decide whether a message needs to be created.

For example, if a voucher is created for \$20,000 for capitalized equipment and the accounting department decides that the controller must be notified of any vouchers entered for greater than \$10,000, the information needed to create a message from voucher entry would be “who is the controller,” and “what are the keys to voucher entry.” At this point, you need to determine what type of J.D. Edwards message would be appropriate and call that type of message within Event Rules. See *Enterprise Workflow Management* for more information about creating complete workflow processes using workflow messages.

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 as titles, such as “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 may also include action messages). These messages supply pertinent information to the user and usually require action be taken.

This section describes the following:

- ☐ Understanding error handling
- ☐ Working with messages
- ☐ Working with the Send Message system function

Understanding Error Handling

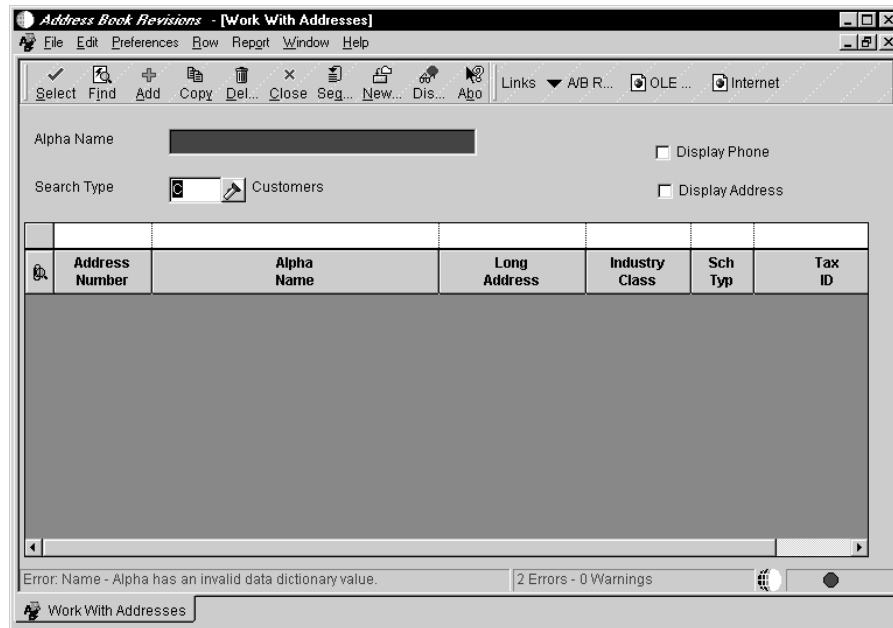
To understand error handling you should become familiar with:

- ☐ The event-driven model
- ☐ Classifications of error settings
- ☐ Resetting errors
- ☐ Multilevel error messaging

Event-Driven Model

Interactive messages display whenever a specific event occurs. This means that errors clear, set and display based on certain events. When an error occurs on 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 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 turns red). In addition, the status bar displays information about the error. To find out details about why the error occurred and how to correct it, the user can press the F8 function key. You can use the F7 function key to walk through the errors on a form if there are two or more errors.



The only way for the user to correct the error is to go into the field again, type in the correct value and leave the field. This causes the same *Control is Exited* event to be run again, which clears the error and sets it again in case an incorrect value is entered again. If the value typed in this time is correct, no error is set.

When is an Error Set?

When a user enters data in a form field or grid cell and tabs out, the runtime engine edits the value that was entered. If the edit fails, (such as a value is not found in the User Defined Code table, or an invalid date has been entered), an error is issued with the appropriate error code and the field is highlighted. The error message and message count are displayed on the status bar.

How Does the Application Know Which Field to Highlight?

When an error is issued, the handle is on the field. If the error is within a grid, then the offending grid cell coordinates (row and column) are identified. OneWorld stores this information in an *Error_Ctrl_Key* structure, along with the application form handle. At the time the error is displayed, the color of the control is changed and the error information is displayed 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 Is an Event ID Used for Error Setting?

OneWorld 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. Validation is done and the error is cleared if no more errors are found on this control.

Error Setting

Errors are set:

- Automatically
- Manually

Automatic Error Setting

The runtime engine validates a field in the following instances:

- The item is a data dictionary item, for which an edit rule trigger has been defined in the data dictionary. In this case, the field for the data item is validated across all OneWorld applications. Data dictionary triggers can also be assigned or redefined using overrides within the Forms 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 things such as 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 the following system functions for setting an error message:

- Set Edit Control Error
- Set Grid Cell Error

You can use Set Edit Control Error to set an error on a Form Control field through event rules. The following parameters must be passed in 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 3/GH951 or a variable)

You can 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 can set an error message using a business function. This is used for manual error setting.

Use one of the following business function APIs to set an error message without text substitution:

- `jdeErrorSet (lpBhvrCom, lpErrInfo, idItem, lpzError)` sets errors through business functions.
- `jdeSetGBRError (lpBhvrCom, lpVoid, (ID)0, "Error#")`

Following is an example of an error message without text substitution:

```
{
    jdeSetGBRError( lpBhvrCom, lpVoid, IDERRcRetainedEarningsLedger_3, "4524");
    idReturn = ER_ERROR;
}
```

The return code successful message has no influence on what happens next. It is used for information.

Use one of the following business function APIs to set an error message with text substitution:

- `jdeErrorSet (lpBhvrCom, lpvoid, idItem, lpzError, lpDs)` sets errors through business functions using substitution text. It always gets called
- `jdeSetGBRErrorSubText`

`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.

Following is an example of the setup for a text substitution error message:

```

/*****
*****
* Declare structures
*****
*****/
DSDE0018    dsDE0018; .
.
.
/*****
*****
* Main Processing
*****
*****/
    strncpy(dsDE0018.szAccountID, (const char
*)(lpDS->szAccountID),
    sizeof(dsDE0018.szAccountID));
.
.
.
if (idReturn != ER_SUCCESS)
{
    jdeSetGBRErrorSubText(lpBhvrCom, lpVoid,
IDERRszAccountID_1, "044E", &dsDE0018);
    JDB_CloseTable(hRequestF0901);
    JDB_FreeBhvr(hUser);
    return ER_ERROR;
}

```

In this example, DSDE0018 is the data structure for the error messages and is declared in jdeapper.h. Jdeapper.h is where all of the data structures are declared for text substituted error messages used by J.D. Edwards. Non-J.D. Edwards substituted error messages should be defined in their own global declaration header file or in the function's header file that is using the structure. The line IDERRszAccountID_1 in this example contains the values to substitute.

A business function may call a validation routine (jdeddValidation), which when called may 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 jdeapperr.h

Resetting Errors

The system will reset errors on the OK button or on the Find button. This 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* - this includes 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

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:

1. Setup *Error_Event_Key* structure with event *Button Clicked*

2. Clear event errors on OK/Select *Button Clicked*
3. 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*
4. 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, suppose 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 and you can find out the company number ID in the second business function.

► To create multi-level error messaging

1. Open the called business function's header file and determine the maximum number of possible mapping fields.

Calling Function's Header File	Called Function's Header File
<code>#define IDERRszComputerid_1 1L</code>	<code>#define IDERRcHeader_1 1L</code>
<code>#define IDERRszCompany_2 2L</code>	<code>#define IDERRcEvent_2 2L</code>
<code>#define IDERRszDate_3 3L</code>	<code>#define IDERRDetail_3 3L</code>
<code>#define IDERRszValue_4 4L</code>	<code>#define IDERRCompany_4 4L</code>
<code>#define IDEERszPoint_5 5L</code>	<code>#define IDERRDateOpen_5 5L</code>

2. Create an Error map section in the calling business function's C file.

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 N1234 1  
*****/
```

The number "1234" is the data structure number of the called business function. The number "1" is the number of mapped fields.

Before each #define statement there should be a comment that refers to the business function name that this number will be used for.

In the variable section create a "cm_xxx" map array for each function that needs to return errors.

For example:

```
/******  
* Variable declarations  
*****/  
  
CALLMAP      cm_1234[N1234]={IDERRCompany_2, IDERRCompany_4};
```

The array is mapped from IDERRCompany_2 to IDERRCompany_4

The function call to jdeCallObject is:

```
idReturn = jdeCallObject("ValidateCompanyName",  
    ValidateCompanyName, lpBhvrCom, lpVoid, &ds1234,  
    cm_1234, N1234, (char *)NULL, (char *)NULL, (int)0);
```

The business function sets the error on the ID field.

Working with Error Messages

OneWorld displays messages automatically based on certain events. For example, you can display an error message whenever an invalid value has been entered into a field.

OneWorld uses the data dictionary glossary to display messages. The message is contained within the glossary portion of a data item. By leveraging the existing framework of the data dictionary, you do not have to create a messaging system from scratch.

This chapter describes the following:

- ☐ Locating an existing error message
- ☐ Creating a simple error message
- ☐ Creating a text substitution error message
- ☐ Attaching an interactive message data item

Locating an Existing Message

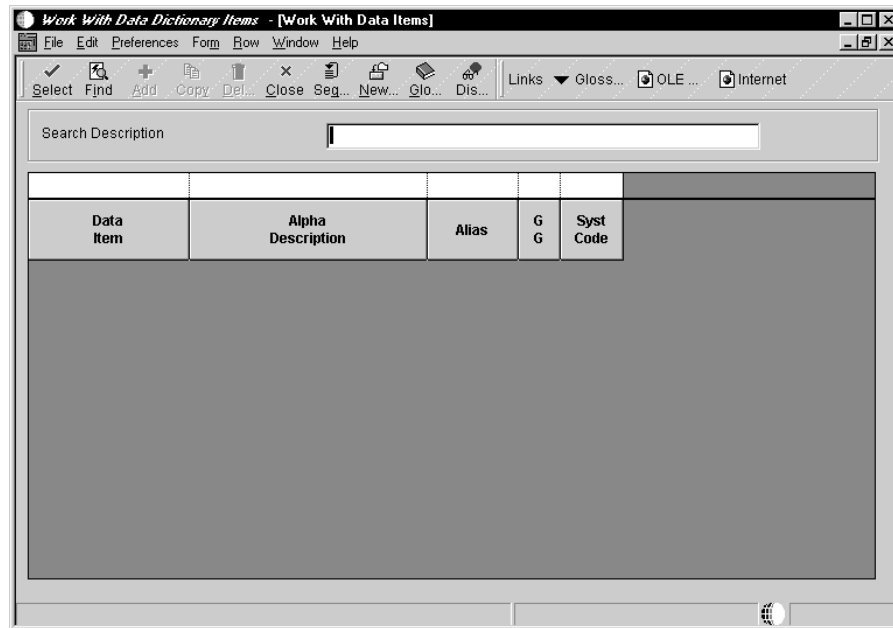
You should use an existing message if possible instead of creating a new one. Use Work with Glossary Items to locate an existing message. You can narrow your search by using glossary group types and descriptions similar to the error you want.



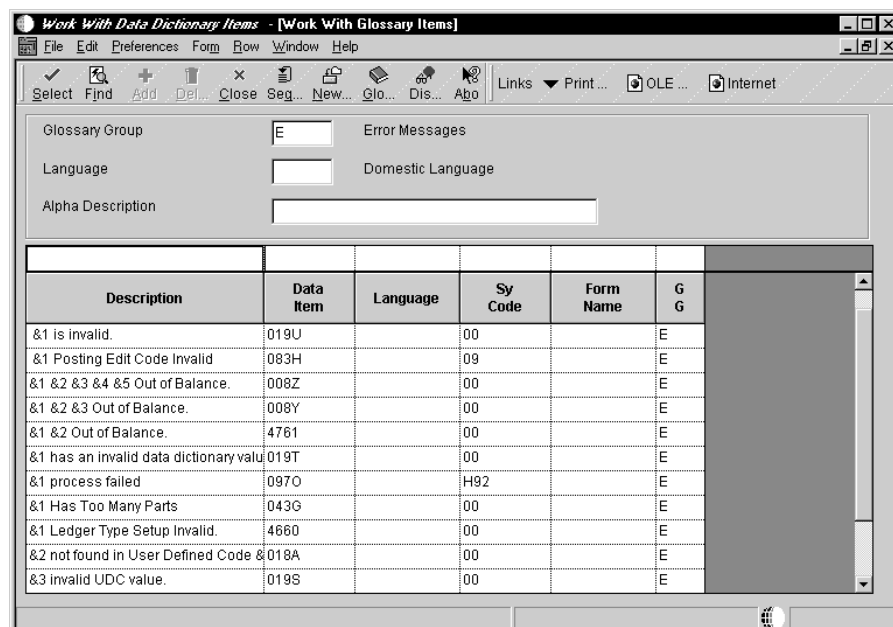
To locate an existing error message

On Work with Data Dictionary Items

1. Choose Glossary Data Item from the Form menu.



2. On Work with Glossary Items choose the message you wish to use.



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 select an existing item to display a message.

Before the error message has functionality, you must connect it to a form control using event rule logic.

► To add an interactive message data item

On Work with Glossary Items

Description	Data Item	Language	Sy Code	Form Name	G

1. From the Form menu, choose Add.

The screenshot shows a software window titled "Work With Data Dictionary Items - [Glossary Header Revisions]". It has a menu bar with File, Edit, Preferences, Form, Window, and Help. Below the menu is a toolbar with icons for OK, Cancel, Glossary, Display, and Abort, along with buttons for Links, Glossary, OLE, and Internet. The main form area contains several input fields and sections:

- Data Item:** 024Y
- Invalid Date:** (text field)
- Glossary Group:** E
- Language:** (text field)
- Domestic Language:** (text field)
- Error Messages:** (text field)
- Form:** (text field)
- System Code:** 42
- Sales Management:** (text field)
- Description:** Invalid Date
- Error Message:**
 - Error Level:** 1
 - Error Message:** (text field)
 - Data Structure:** (checkbox, currently unchecked)

2. On Glossary Header Revisions, complete the following fields:

- Data Item
- Description
- Glossary Group
- System Code
- Error Level

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 some values that vary.

For example, suppose 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 xx in the message will be replaced by the search type the user entered.

The following tasks are required to create a text substitution error message:

- ☐ Adding an interactive message data item
- ☐ Defining the glossary text for a runtime message
- ☐ Attaching a data structure template to a message

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.

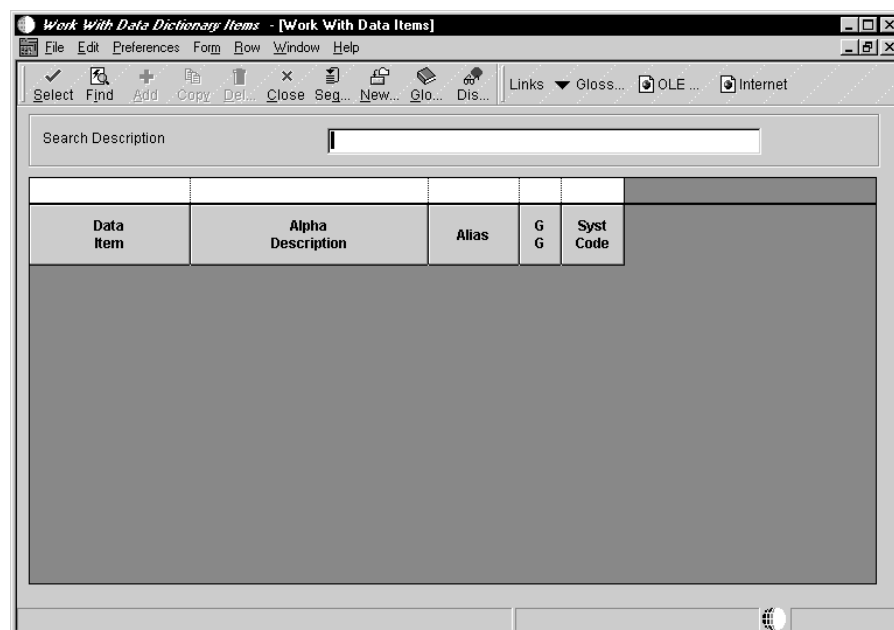
Adding an Interactive Message Data Item

You can create a new data item or select an existing item to display a message. This example creates an error message in the Data Dictionary for error ID 018A.

Before the error message has functionality, you must connect it using event rule logic.

► To add an interactive message data item

On Work with Data Dictionary Items



1. From the Form menu, choose Glossary Data Item.

2. On Work with Glossary Items, choose Add from the Form menu.

3. On Glossary Header Revisions, complete the following fields:

- Data Item
- Description
- Glossary Group
- System Code
- Error Level

When you type the description, use & followed by a number for each subset variable. In this example, the short description is &2 *not found in User Defined Code* &3 &4.

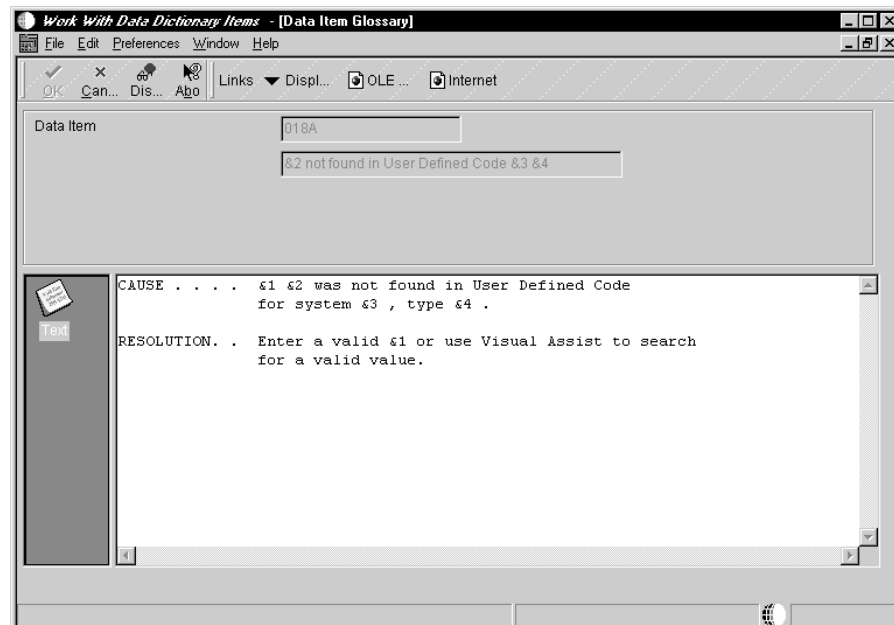
Now you are ready to define the glossary text to display at runtime.

Defining the Glossary Text for a Runtime Message

► To define the glossary text for a runtime message

On Work with Glossary Items

1. Choose Glossary from the Form 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. Put a period with no blank spaces at the end of your glossary so you do not have any repetition problems.

The order of the data structure is the index of the &x. If the structure contains four members, assign &1 to the first member, &2 to the second, and so on.

Description = &1

UDC Value = &2

System = &3

Type = &4

3. When your text is complete, click OK.

OneWorld returns to the Glossary Header Revisions form.

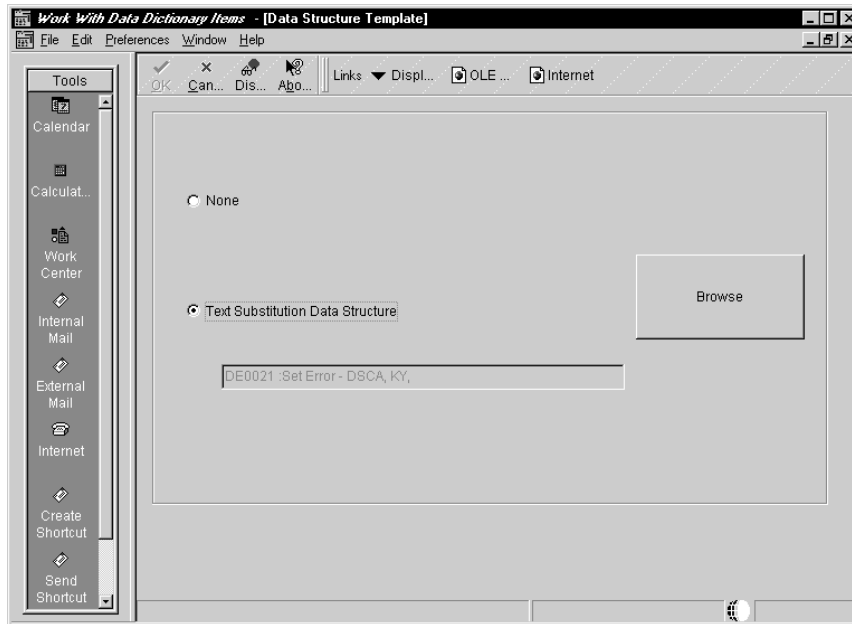
Attaching a Data Structure Template to a Message

Each text substitution error message requires a corresponding data structure. If the data structure does not already exist, then you must create it. Make sure the members in your message match the members in your data structure. See *Data Structures*.

► To attach a data structure template to the message

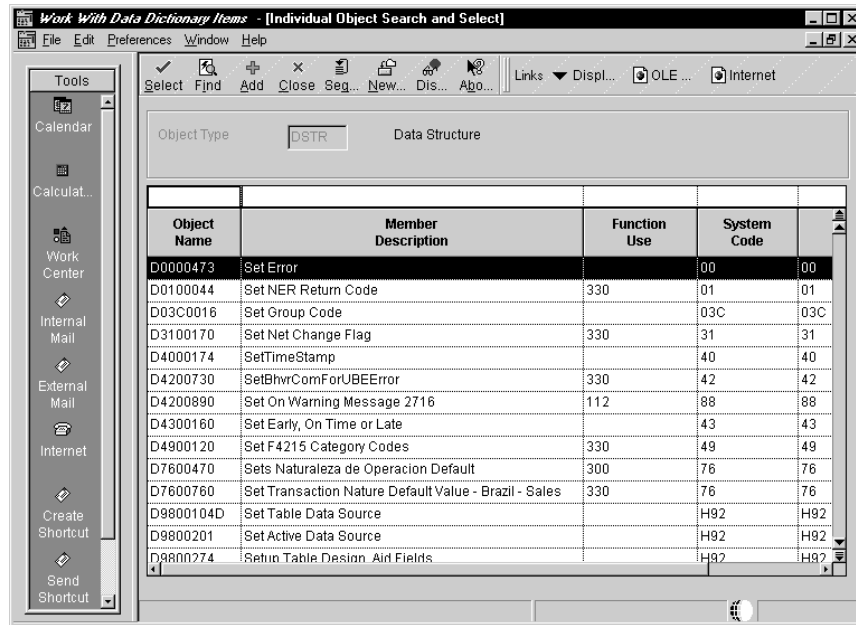
On Glossary Header Revisions

1. From the Form menu, choose Data Structure.



You can browse to locate an existing structure. However, if there is not an existing structure that meets your needs for the interactive message, you must create one.

2. To locate an existing structure click the Browse button.



- On Object Search, locate the data structure and click OK.

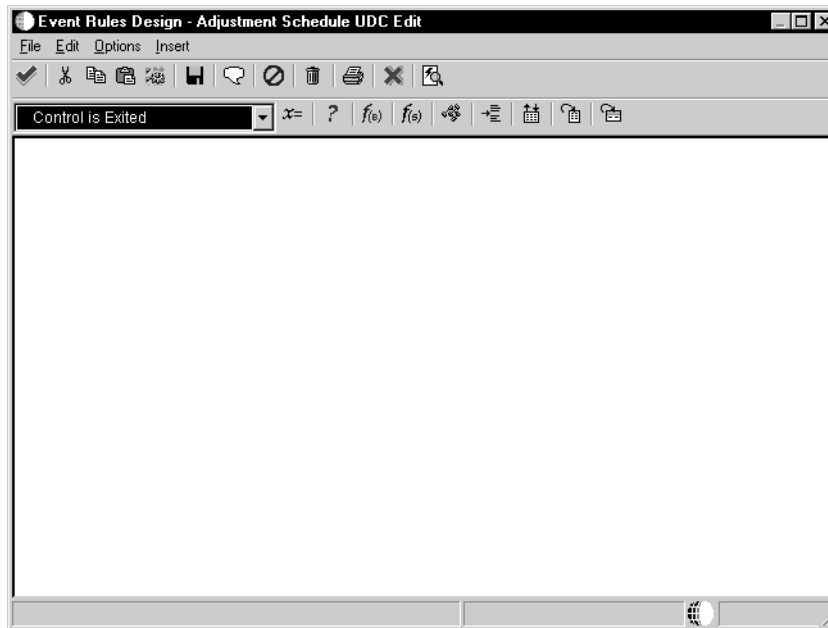
Attaching an Interactive Message Data Item

To make the error message display at runtime, you must attach a system function.

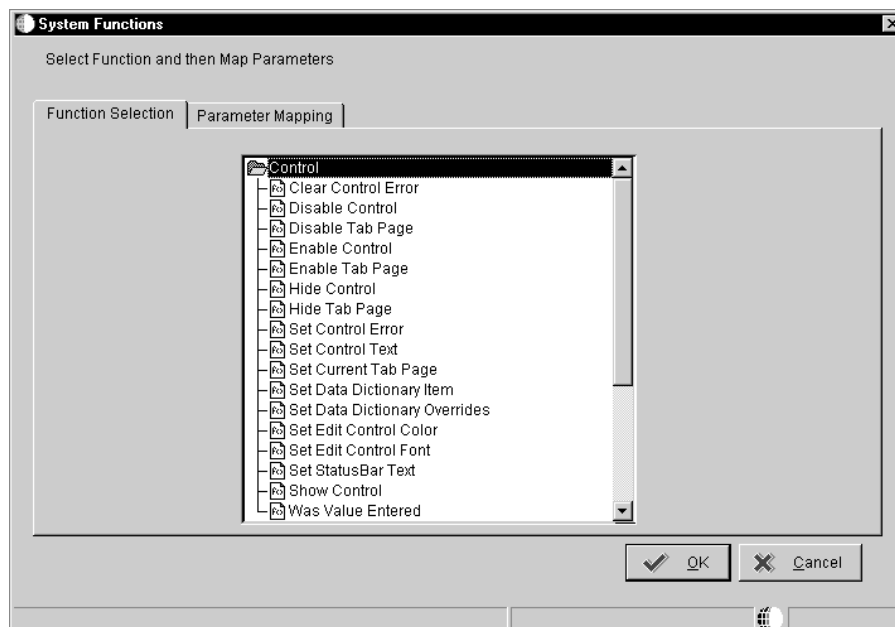
► To attach a message data item

On Event Rules Design

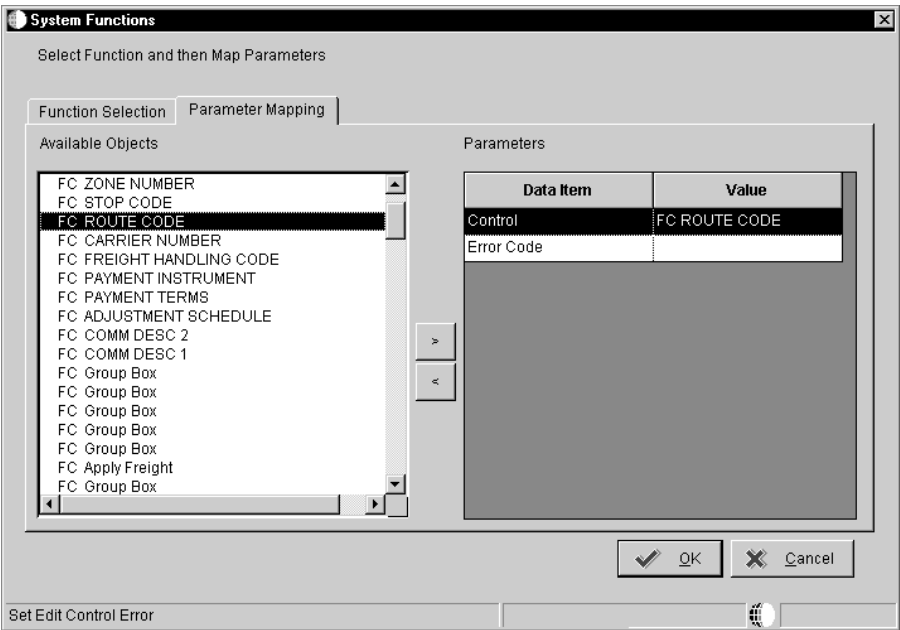
- Click the system function button.



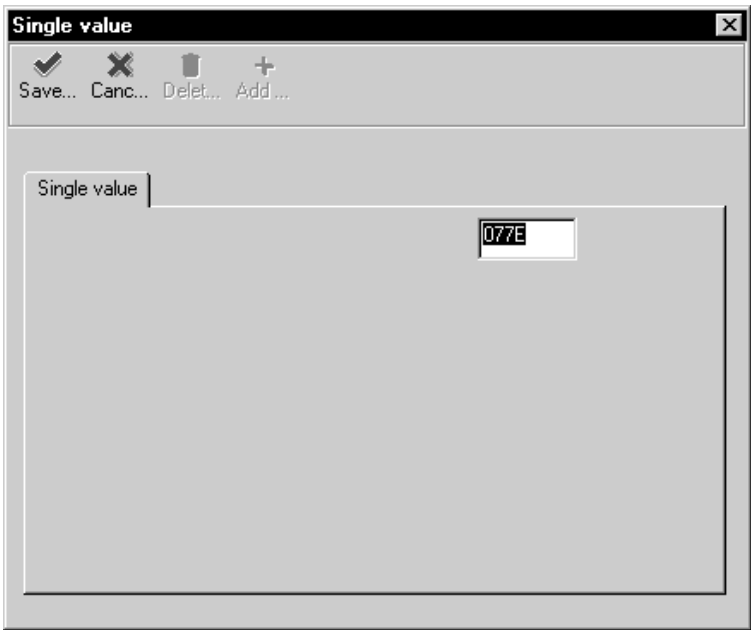
2. Select the system function you wish to use.



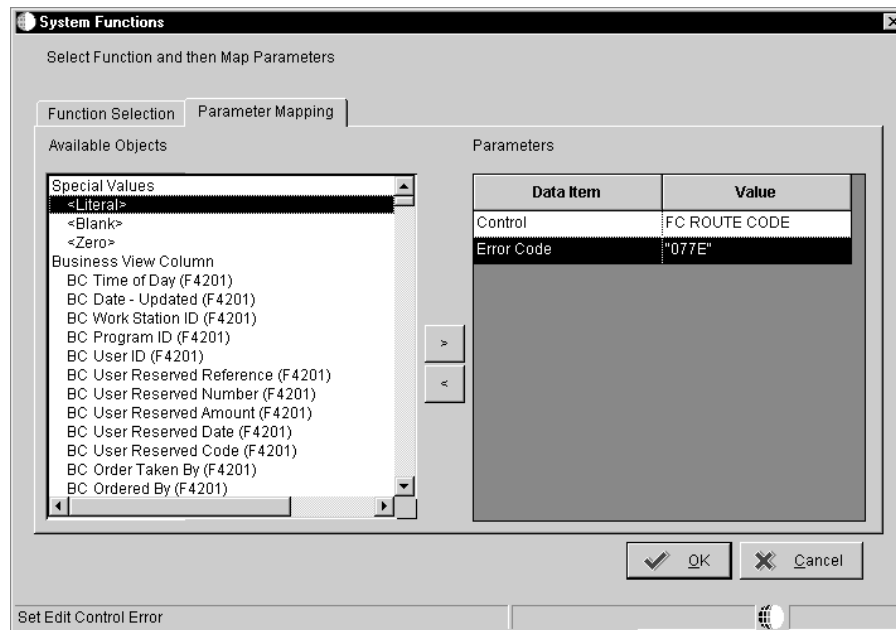
3. Use available objects to complete the control value in Parameters.



4. Choose <Literal> for the Error Code Value in Parameters.



Use the number of the data dictionary item that contains the error message you want to appear for the literal value.



Working with the Send Message System Function

The Send Message API involves the following tables:

- F01131 - Message Header table
- F01131M - Multi-level Message
- F01133 - Detail table
- F00165 - Media Object Storage

The Message Header table (F01131) and the Message Detail table (F01133) are used by the Message Center application for delivery of messages. They contain information on 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 with, 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 comprised 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
Recipient	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 will be used.	NO “None” should be selected if a message template is used.
*Text	The main body of the message.	The message that will be sent, or Blank if a message template will be used.	NO “None” should be selected if a message template is used.
Active	This indicates whether there should be a connection between the message being sent and another form.	When this item is selected the user will be 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	This is 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 onto the Subject and Text fields.

The following example illustrates how to use the send message system function.



To use the send message system function

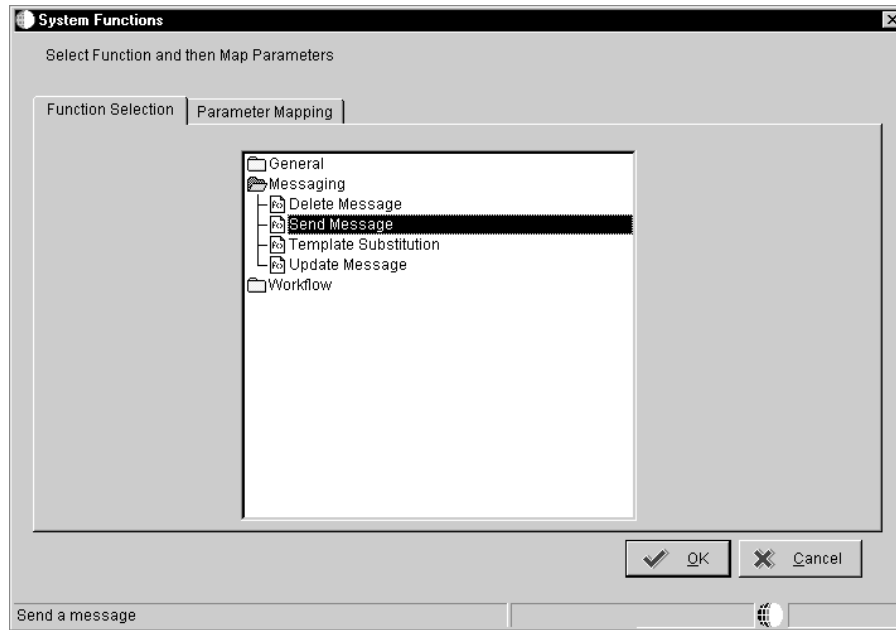
On Object Librarian

1. Choose the table you wish to use.

Location	Path Code	S C	SAR	User	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1836424	SO828	5/6/98	15:41:36	
INTELNTC	PROD	1	1836424	SO828	5/6/98	15:42:00	
PETERSONP2	APPL_PGF	3	1	MP233741	8/3/98	08:46:46	
PRATTET	APPL_PGF	3	1836424	TP5821385	7/30/98	10:50:45	

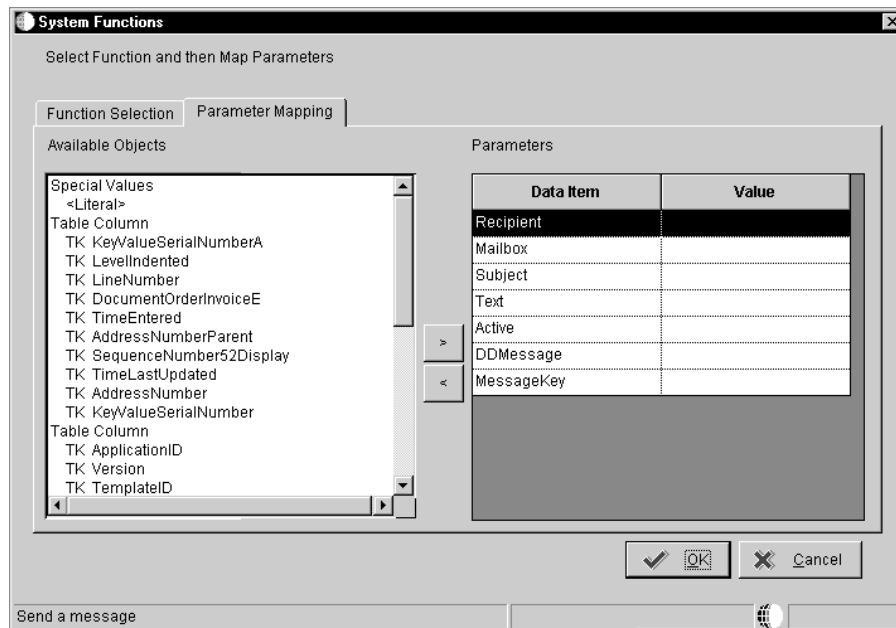
- On Table Design Aid, from the Form menu, choose Table ER.

- Click System Functions



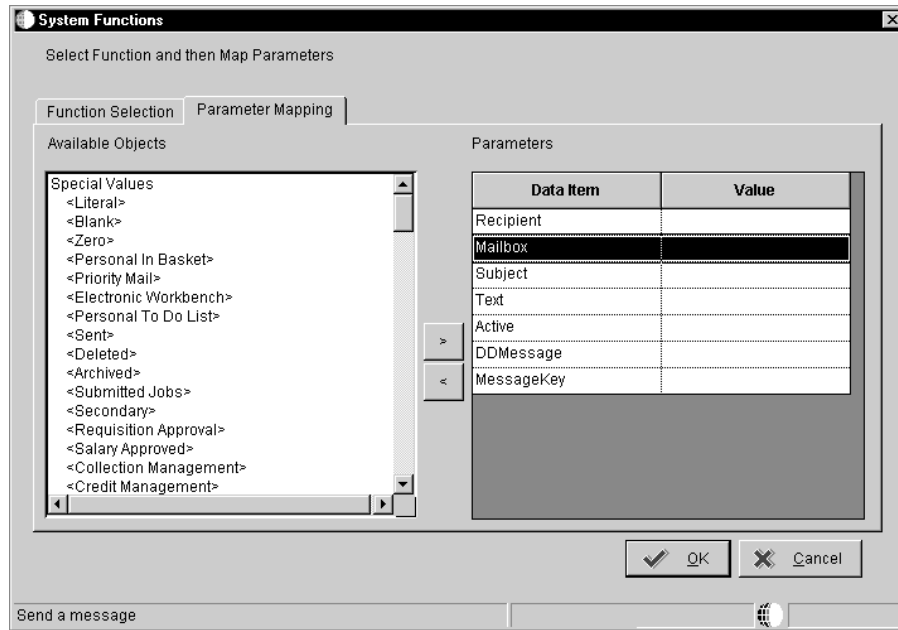
4. Designate who is to receive the message in the Recipient field.

This field typically uses an address book number. You can also use an e-mail address here.



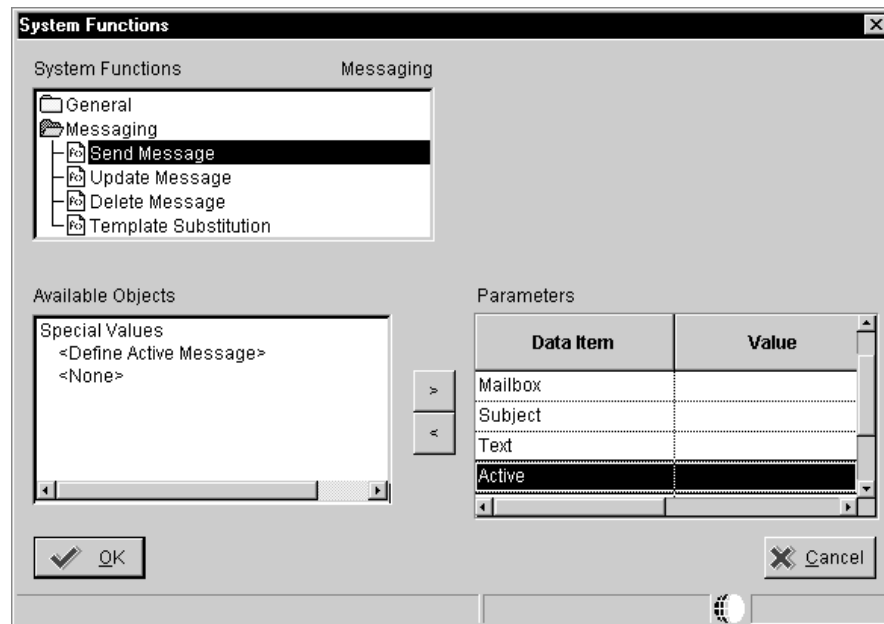
5. Next, designate the Mailbox.

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 will be displayed when focus is on the Mailbox field. The following form is an example of what might be displayed for mailboxes.



- Choose Define Active Message to begin the form interconnection process.

The active message portion of the message works like form interconnections from event rules.

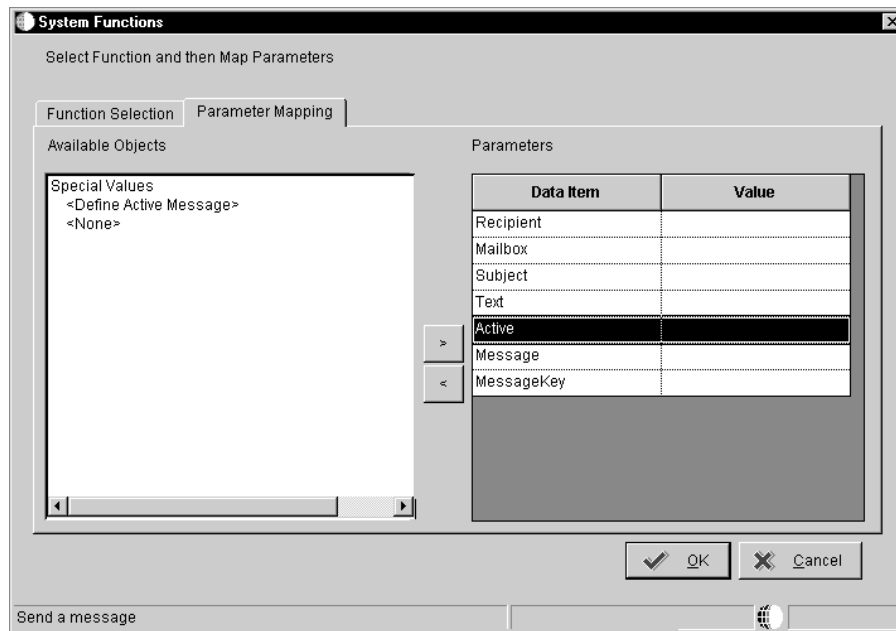


Defining an Active Message

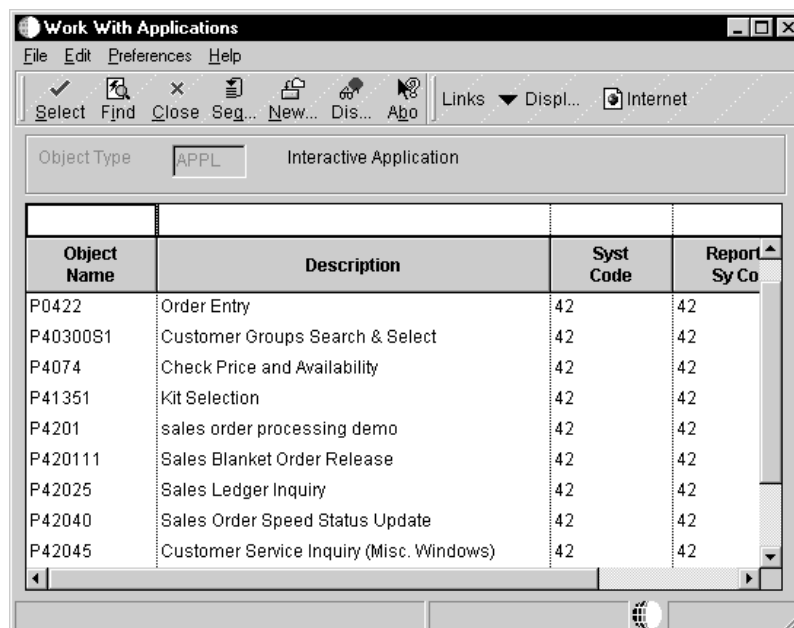
The following forms illustrate what forms appear as you define an active message.

► To define an active message

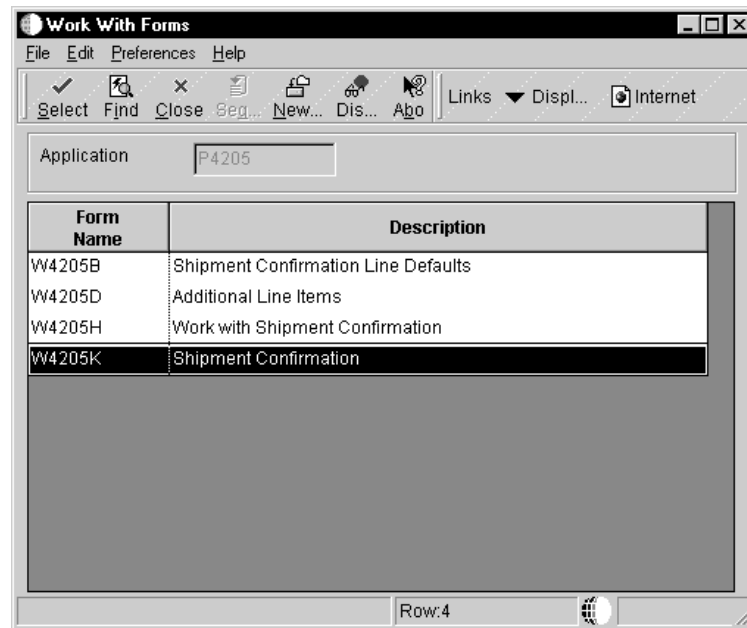
1. Double-click on Define Active Message.



Work with Applications appears.



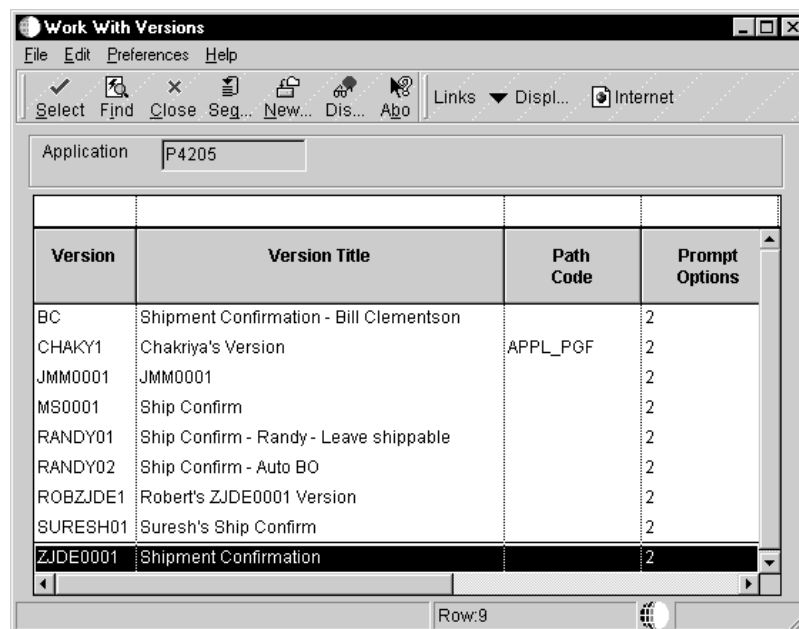
2. Choose the application you wish to work with.



Form Name	Description
W4205B	Shipment Confirmation Line Defaults
W4205D	Additional Line Items
W4205H	Work with Shipment Confirmation
W4205K	Shipment Confirmation

Row: 4

- Choose the form you wish to work with.

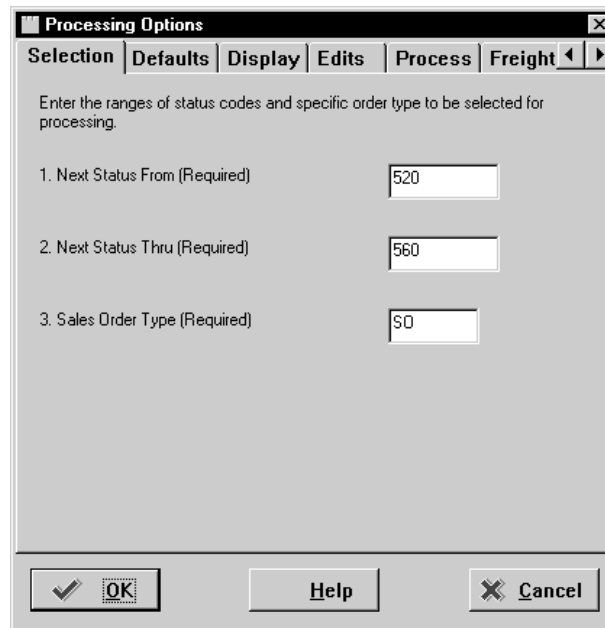


Version	Version Title	Path Code	Prompt Options
BC	Shipment Confirmation - Bill Clementson		2
CHAKY1	Chakriya's Version	APPL_PG	2
JMM0001	JMM0001		2
MS0001	Ship Confirm		2
RANDY01	Ship Confirm - Randy - Leave shippable		2
RANDY02	Ship Confirm - Auto BO		2
ROBZJDE1	Robert's ZJDE0001 Version		2
SURESH01	Suresh's Ship Confirm		2
ZJDE0001	Shipment Confirmation		2

Row: 9

- Choose the version you wish to work with.

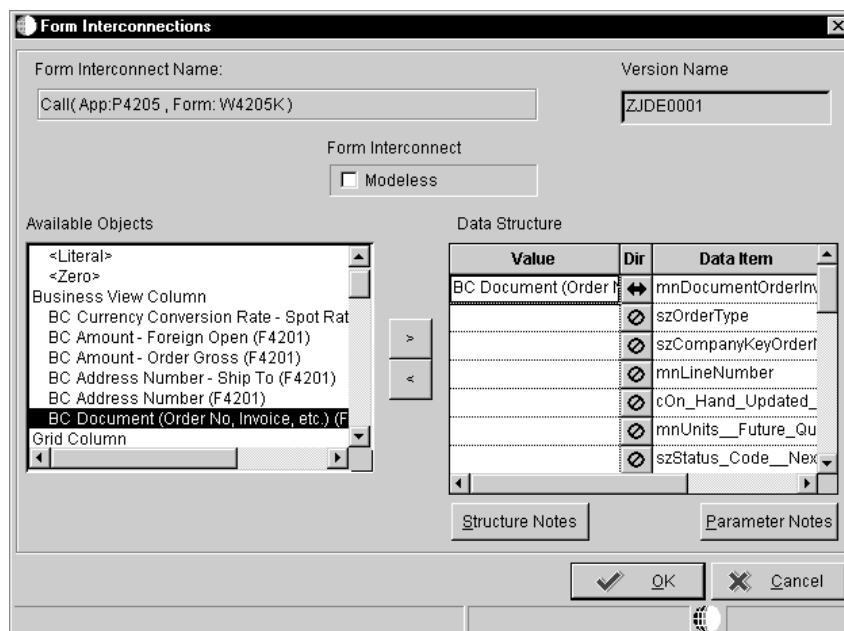
The Processing Options form may appear if there are processing options attached to the form.



The **Processing Options** dialog box has tabs for Selection, Defaults, Display, Edits, Process, and Freight. The Selection tab is active, showing instructions to enter ranges of status codes and specific order type. It contains three required fields: 'Next Status From' (520), 'Next Status Thru' (560), and 'Sales Order Type' (SO). Buttons for OK, Help, and Cancel are at the bottom.

Field	Value
1. Next Status From (Required)	520
2. Next Status Thru (Required)	560
3. Sales Order Type (Required)	SO

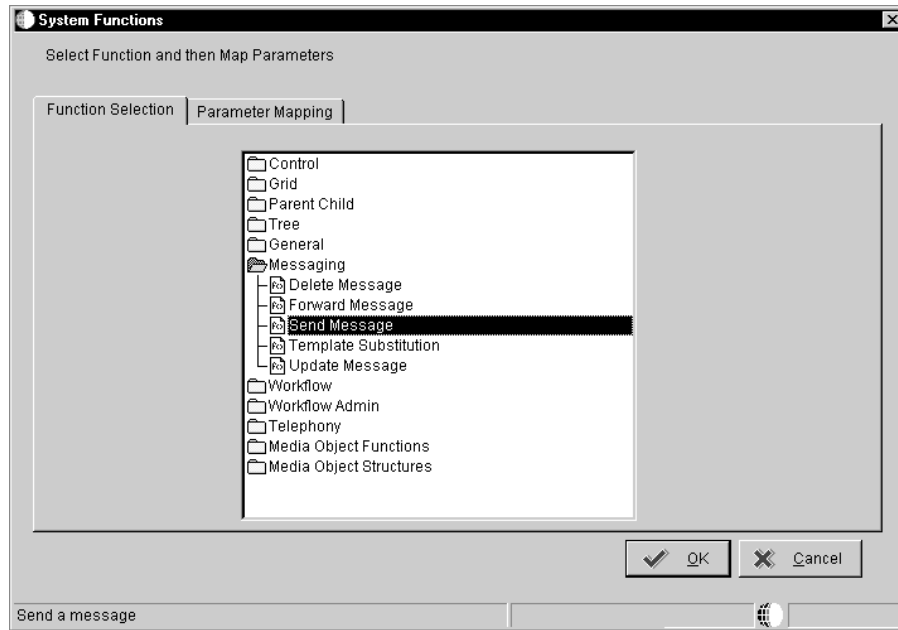
The Form Interconnections form displays, allowing assignment of all the values to be passed.



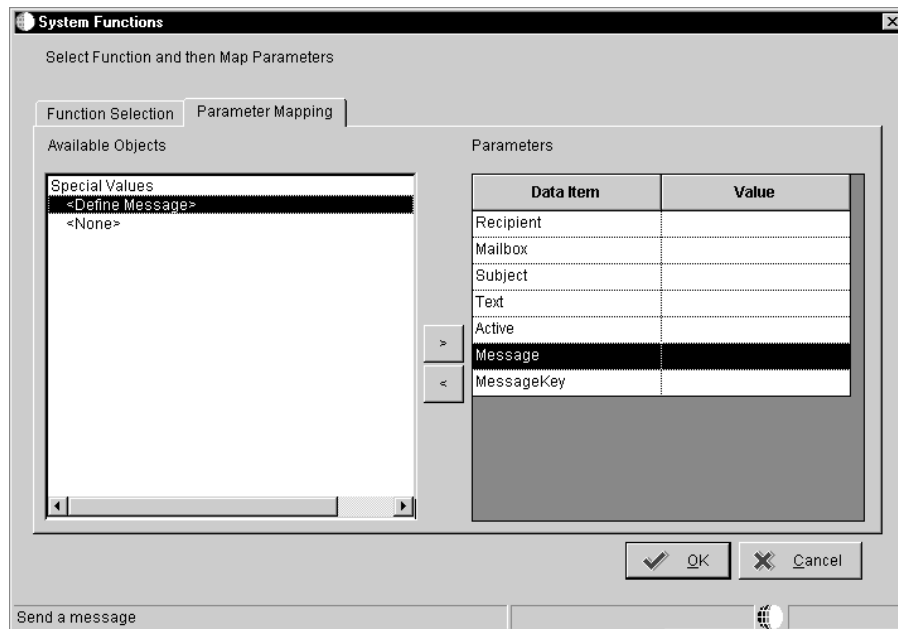
The **Form Interconnections** dialog box shows the 'Form Interconnect Name' as 'Call(App:P4205, Form:W4205K)' and 'Version Name' as 'ZJDE0001'. It includes a 'Form Interconnect' checkbox for 'Modeless'. The 'Available Objects' list includes '<Literal>', '<Zero>', and various Business View Columns. The 'Data Structure' table maps values to data items.

Value	Dir	Data Item
BC Document (Order No, Invoice, etc.) (F4201)	↔	mnDocumentOrderIn
	⊗	szOrderType
	⊗	szCompanyKeyOrderf
	⊗	mnLineNumber
	⊗	cOn_Hand_Updated_
	⊗	mnUnits__Future_Qu
	⊗	szStatus_Code__Nex

5. Designate the values you wish to pass.



6. On System Functions select Define Message.



You will be prompted for the selection of the message.

7. Enter the appropriate alias in the Dictionary Item field and do a Find.

Text Substitution

Dictionary Item:

Description:

Glossary:

Available Objects

- <Literal>
- Business View Column
- BC Order Taken By (F4201)
- BC Currency Conversion Rate - Spot Rate
- BC Date - Cancel (F4201)
- BC Date - Actual Ship Date (F4201)
- BC Date - Original Promised Delivery (F4201)

Parameters

Subst	Value

The substitution points are numbered &1, &2, etc. The parameter numbers correspond to the substitution numbers. Substitution variables can appear in both the Description and the Glossary of the message.

Text Substitution

Dictionary Item:

Description:

Glossary:

Available Objects

- <Literal>
- Business View Column
- BC Order Taken By (F4201)
- BC Currency Conversion Rate - Spot Rate
- BC Date - Cancel (F4201)
- BC Date - Actual Ship Date (F4201)
- BC Date - Original Promised Delivery (F4201)

Parameters

Subst	Value

- Map the appropriate objects to the substitution parameters so that the message will be complete.



Batch Error Messages

The error message system gives end users a consistent interface to review errors when working with OneWorld batch programs. When a batch program has finished processing all messages regarding the success or failure of the job, a message is sent to the end user using the Work Center. To make these messages useable, a tree (or parent/child) structure is used 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 a form by clicking on the message.

This section describes the following:

- ☐ Understanding batch error messaging
- ☐ Creating a level-break message



Understanding 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 what possible messages OneWorld users will need. You might create a number of different messages that are generated when a journal entry report is run. For example, you might create a single message stating that the report completed normally if the report balances. Or, you might create multiple levels of messages describing errors if the report is out of balance. The first level might state that the report completed with errors and additional additional levels of errors would then explain the specifics of the error.

Level-Break Messages

A level-break message is a message you can create 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, or nonaction messages, which can be messages that instruct the user to review some information.

Understanding the role of a level-break message requires taking a closer look at how errors are created in batch jobs. Most errors are created by edits that occur at level breaks in the entire batch process. It is at these different level breaks that you want to group together any errors that may have occurred. The mechanism that groups these errors together is the level-break message. This implies that each reporting program that uses the Work Center application program interface (API) to manage errors will need to create one level-break message for each phase of level-break.

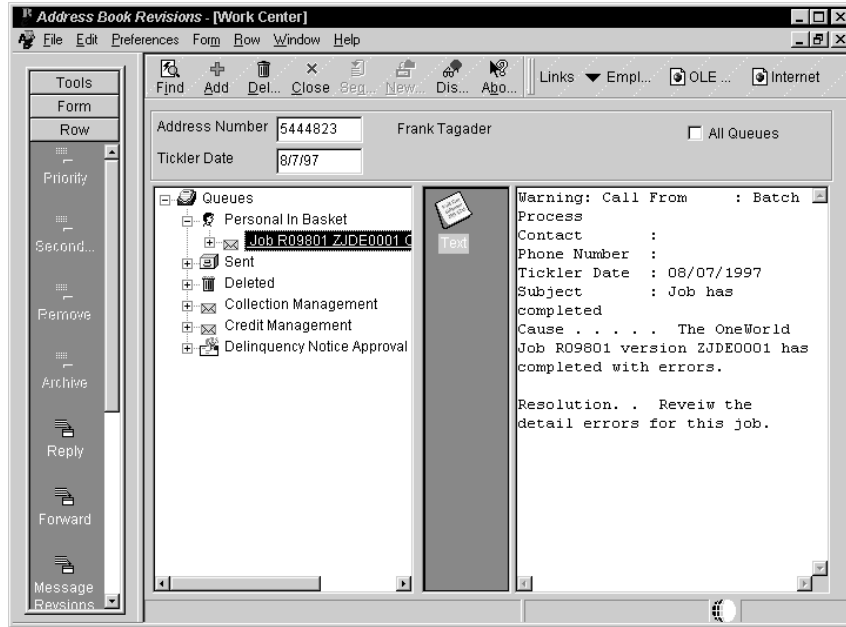
See Also

- *Understanding Level Breaks* in the *Enterprise Report Writing Guide* for more information about level breaks within sections.

The following examples show the how messages might appear when an out-of-balance journal entry upload has completed.

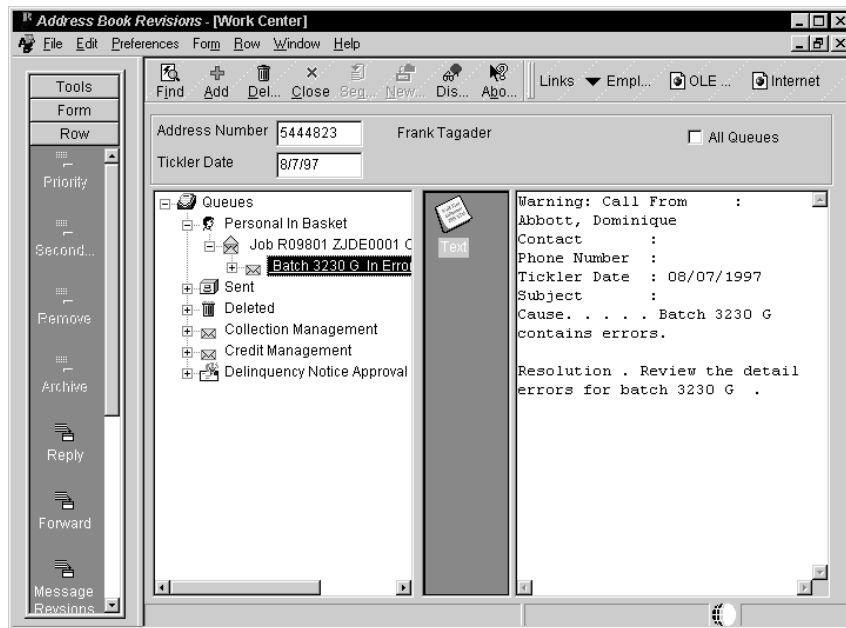
First-Level Messages

First-level messages appear when users open their personal in baskets. The plus symbol next to the message indicates that there are additional levels 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 detail of the errors.



Second-Level Messages

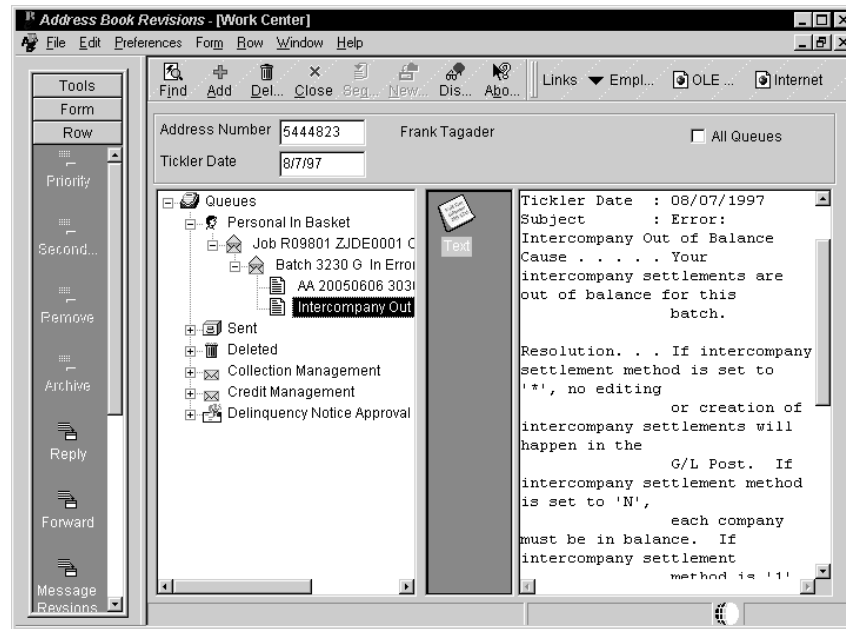
Second-level messages appear when the user double-clicks the plus sign next to the first-level message. In this example, the second-level messages informs the user the batch number that needs to be reviewed.



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 instructs the user that

the batch job completed with errors because it was out of balance, and then offers several solutions that will fix the problem.



Text Substitution Error Messages

Any error messages that you write need to 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 that will be substituted at run time. You set up text substitution messages using the data dictionary, which helps ensure consistency of jargon and terms used. 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 kept 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 *Error Handling* section of this guide for procedures on 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

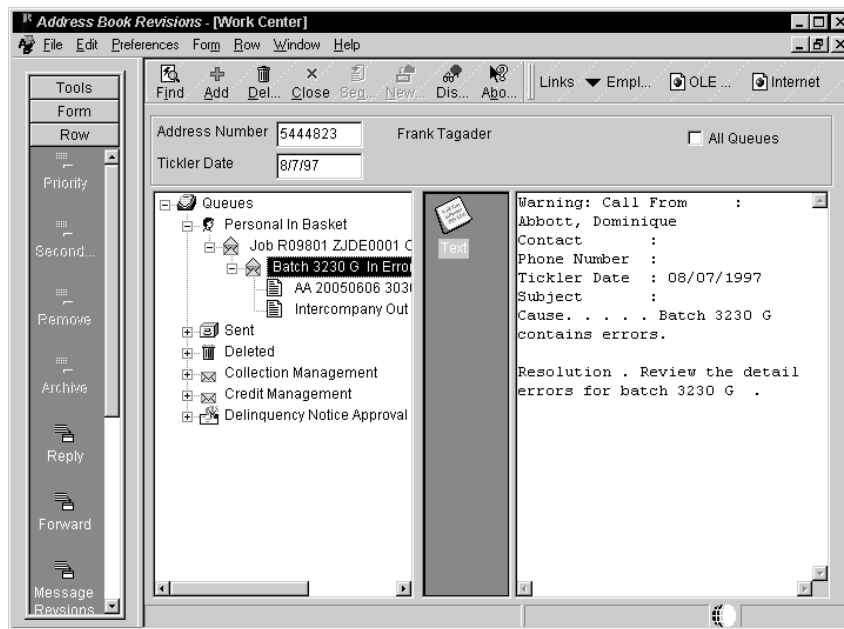
After a user reviews errors and decides what needs to be fixed, the user usually needs to go to a revision program to fix the errors. The Work Center allows you to set up

level-break messages that will give the user the ability to fix the errors by calling the corresponding revision program directly from the Work Center. The term “action message” refers to the ability to call a OneWorld application and to pass the variables needed to that application. For example, the user would be able to automatically load the OneWorld Voucher Revision form with the record in error by clicking a shortcut within the error message. It is up to you to call the appropriate application and pass the correct values to that application. Note that in some cases it may not make sense to call a particular program for a given level. Action messages are highlighted in the grid to differentiate them from nonaction messages.

Understanding How Level Break Messages Work

Level break and action messages are used to group together (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 which batch, document, line, and so on the error occurred. You set these messages using the `jdeSetGBRError` or the `jdeSetGBRErrorSubText` function.



In this 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 pieces. The first piece of a level-break message is the text for the message, and the second indicates whether the message will be an action message.

All level-break messages contain the text piece, 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. It is at these level breaks that the Work Center API should be called. Usually a level break should be set up any time 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.

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 run time 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 OneWorld 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. Most likely the reporting program 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 most likely be sending level-break numbers in descending level-break order.

For example, the actual series of level-break calls to the API might look like this: 4,4,4,3,4,4,3,2,4,3,2,1.

This series shows that the call structure started four levels down. The first call at level 4 would let the Work Center API find any messages that have occurred at that time and create child messages using the level 4 message as the parent. Note that if no errors occurred, then no messages would have been created. This call sequence example shows that the API was called at a level 3 after three calls to level 4. When the call to level 3 is made, the Work Center API remembers if any level 4 messages have been written or not. In other words, if no errors occurred when any of the level 4 calls were made, then the Work Center API will not create the level 3 message. If there was at least one error at any of the level 4 calls, the level 3 and the level 2 message will be created.

You must call the Work Center API at every level. As explained earlier, the Work Center error messages are created based on a parent/child structure. Therefore, if a level call is skipped, the API has no way to group the child messages and child levels 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, there is no call to level 5.

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 and issues the “job completed” message. The level 1 call to the Work Center API is essential. A level 1 call to the API will not only ensure that no orphan Work Center records are created, but will also clean up all allocated storage used by the Work Center system. The level 1 call to the API should only occur once in the report. Generally it is done 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 active messages or nonactive messages, as explained in *Understanding Batch Error Messaging*.

You create level-break messages by creating data dictionary items, error data structures, business function error structures, and business functions. For additional information about these topics, refer to the related sections in this guide.

This section describes the following:

- ☐ Creating a data dictionary item for a level-break message
- ☐ Creating a data structure for the data dictionary item
- ☐ Creating a level-break message business function data structure
- ☐ Creating a level-break business function
- ☐ Calling the Work Center initialization API
- ☐ Calling the processing Work Center APIs
- ☐ Terminating the process

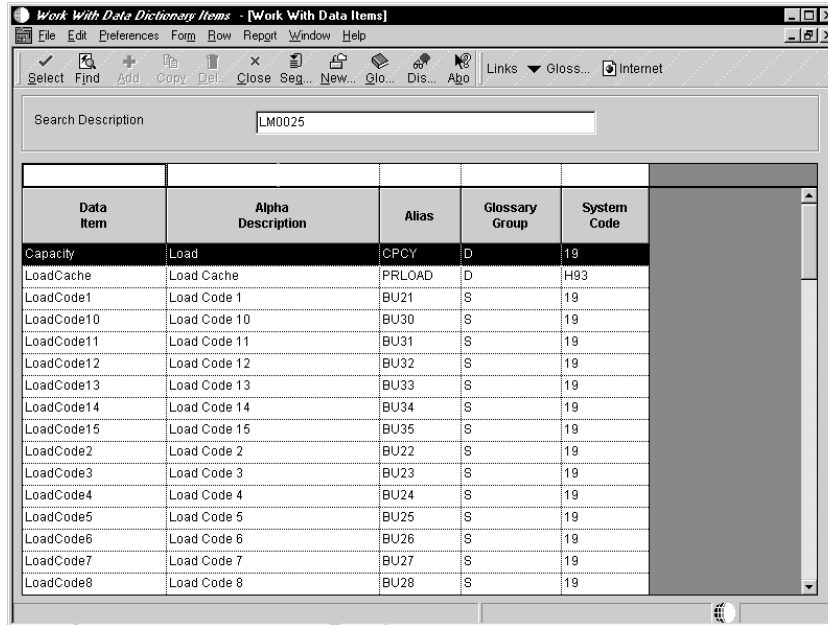
Creating a Data Dictionary Item for a Level-Break Message

The data dictionary item is the text portion of the level-break message. Before you create a data dictionary item, you may want to review the level-break messages that have already been created. It is possible that the level-break message that you want to use already exists. This is the message you see in the workcenter.



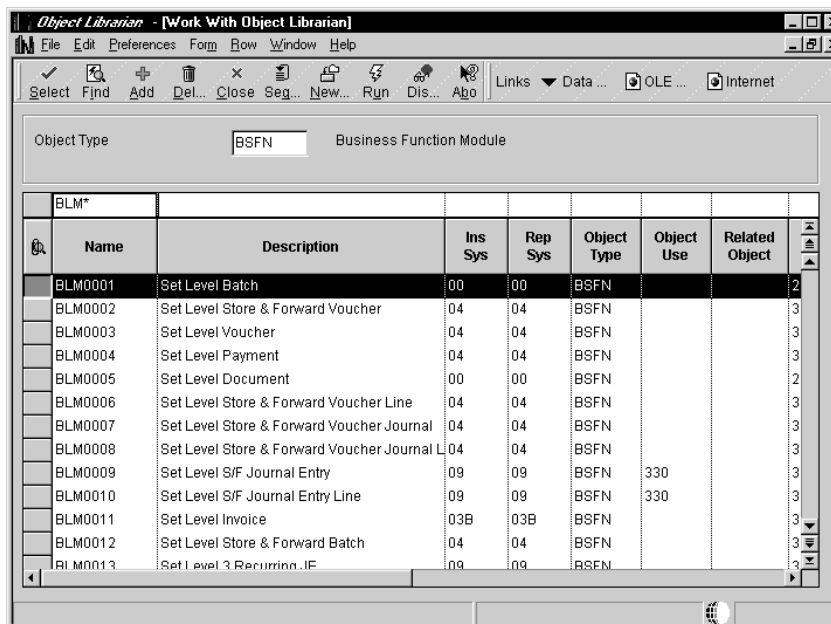
To create a data dictionary item for a level-break message

1. To search through the list of existing level-break messages, use the Object Librarian.



Data Item	Alpha Description	Alias	Glossary Group	System Code
Capacity	Load	CPCY	D	19
LoadCache	Load Cache	PRLOAD	D	H93
LoadCode1	Load Code 1	BU21	S	19
LoadCode10	Load Code 10	BU30	S	19
LoadCode11	Load Code 11	BU31	S	19
LoadCode12	Load Code 12	BU32	S	19
LoadCode13	Load Code 13	BU33	S	19
LoadCode14	Load Code 14	BU34	S	19
LoadCode15	Load Code 15	BU35	S	19
LoadCode2	Load Code 2	BU22	S	19
LoadCode3	Load Code 3	BU23	S	19
LoadCode4	Load Code 4	BU24	S	19
LoadCode5	Load Code 5	BU25	S	19
LoadCode6	Load Code 6	BU26	S	19
LoadCode7	Load Code 7	BU27	S	19
LoadCode8	Load Code 8	BU28	S	19

For every level-break message created, there is a data dictionary item and at least one business function created to reference that item. To find out what level-break messages already exist, use the query by example search to locate all business functions (BSFN) that start with the first three letters, BLM.

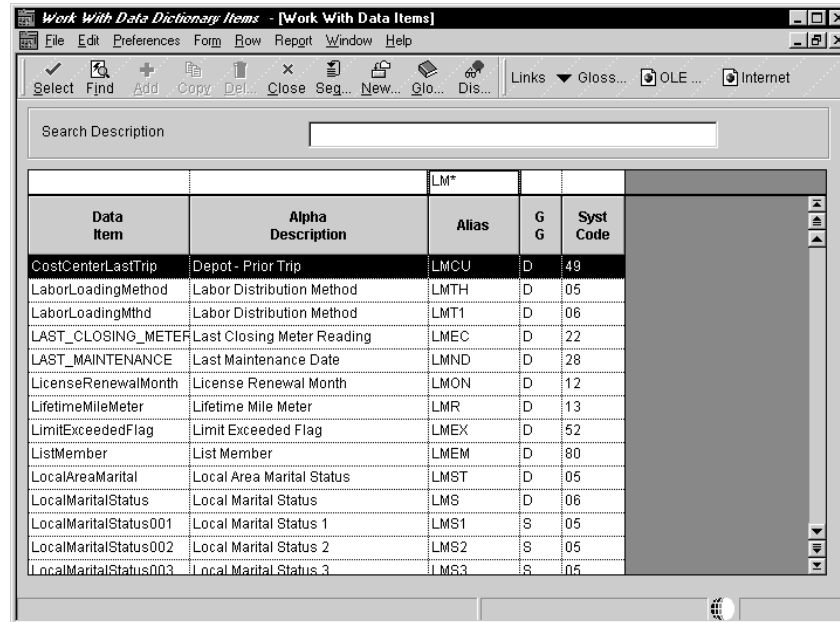


Name	Description	Ins Sys	Rep Sys	Object Type	Object Use	Related Object
BLM*						
BLM0001	Set Level Batch	00	00	BSFN		2
BLM0002	Set Level Store & Forward Voucher	04	04	BSFN		3
BLM0003	Set Level Voucher	04	04	BSFN		3
BLM0004	Set Level Payment	04	04	BSFN		3
BLM0005	Set Level Document	00	00	BSFN		2
BLM0006	Set Level Store & Forward Voucher Line	04	04	BSFN		3
BLM0007	Set Level Store & Forward Voucher Journal	04	04	BSFN		3
BLM0008	Set Level Store & Forward Voucher Journal L	04	04	BSFN		3
BLM0009	Set Level S/F Journal Entry	09	09	BSFN	330	3
BLM0010	Set Level S/F Journal Entry Line	09	09	BSFN	330	3
BLM0011	Set Level Invoice	03B	03B	BSFN		3
BLM0012	Set Level Store & Forward Batch	04	04	BSFN		3
BLM0013	Set Level 3 Recurring JE	09	09	BSFN		3

The BLM naming convention refers specifically to J.D. Edwards-created level-break message business functions. J.D. Edwards recommends naming your own level-break message business functions using the following criteria: BLMxxyy where the xx refers to the system code you are using (between 50 and 55) and the yy is a unique number. This unique number should be carried forward to other items. The maximum number of characters is eight.

NOTE: Use the description field to help you decide which existing messages may work in your report.

If you find several level-break message business functions that seem to fit your report design, write them down and verify the text on the data dictionary item itself, by using the data dictionary and inquiring on data items LM followed by the same numbers that followed the corresponding BLM function name.



Data Item	Alpha Description	Alias	G	Syst Code
CostCenterLastTrip	Depot - Prior Trip	LMCU	D	49
LaborLoadingMethod	Labor Distribution Method	LMTH	D	05
LaborLoadingMthd	Labor Distribution Method	LMT1	D	06
LAST_CLOSING_METER	Last Closing Meter Reading	LMEC	D	22
LAST_MAINTENANCE	Last Maintenance Date	LMND	D	28
LicenseRenewalMonth	License Renewal Month	LMON	D	12
LifetimeMileMeter	Lifetime Mile Meter	LMR	D	13
LimitExceededFlag	Limit Exceeded Flag	LMEX	D	52
ListMember	List Member	LMEM	D	80
LocalAreaMarital	Local Area Marital Status	LMST	D	05
LocalMaritalStatus	Local Marital Status	LMS	D	06
LocalMaritalStatus001	Local Marital Status 1	LMS1	S	05
LocalMaritalStatus002	Local Marital Status 2	LMS2	S	05
LocalMaritalStatus003	Local Marital Status 3	LMS3	S	05

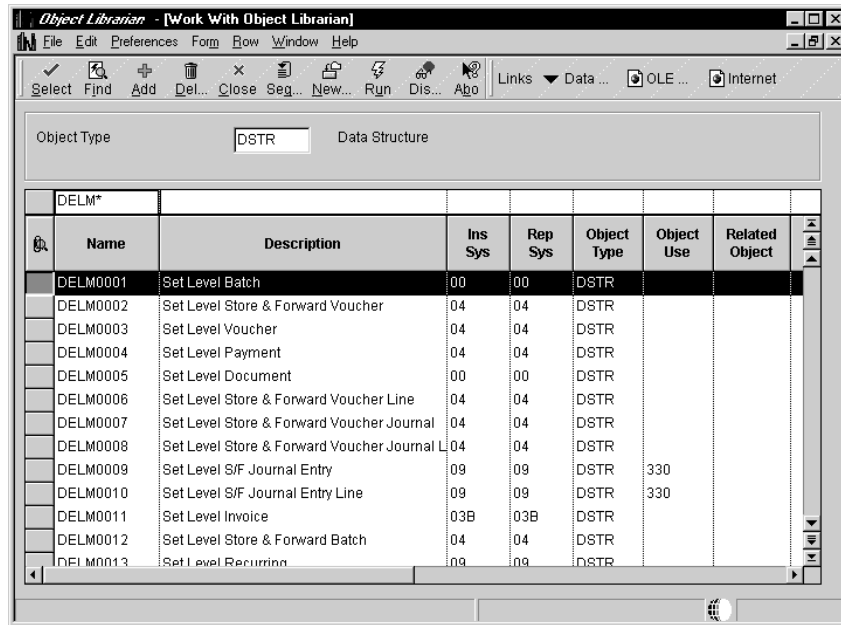
The LM naming convention refers specifically to J.D. Edwards-created level-break messages with text substitutions. These are glossary group Y. J.D. Edwards recommends naming your own level-break message with text substitutions using the following criteria: Lxxyy where xx refers to the system code you are using (between 50 and 55) and the yy is a unique number. The maximum number of characters is eight.

For example, if business function BLM0025 is a level-break message that you want to see the description for, inquire on data item LM0025.

- If the text description meets your criteria, verify any substituted variables that the level-break message uses.

If the text in the LM message contains substituted variables (such as “&1”), you need to verify the data items of the data structure used by the level-break message. When you verify the data structure, it is important that data items in the structure are the exact data items that you will be passing in your report.

To verify that the data item is in the data structure, use the Object Librarian and inquire on data structure DELM followed by the same numbers used before.



The DELM naming convention refers specifically to J.D. Edwards-created level-break message data structures. J.D. Edwards recommends naming your own level-break messages using the following criteria: DELM $xxyy$ where xx refers to the system code you are using (between 50 and 55) and the yy is a unique number. This unique number should be the same number used previously. The maximum number of characters is eight.

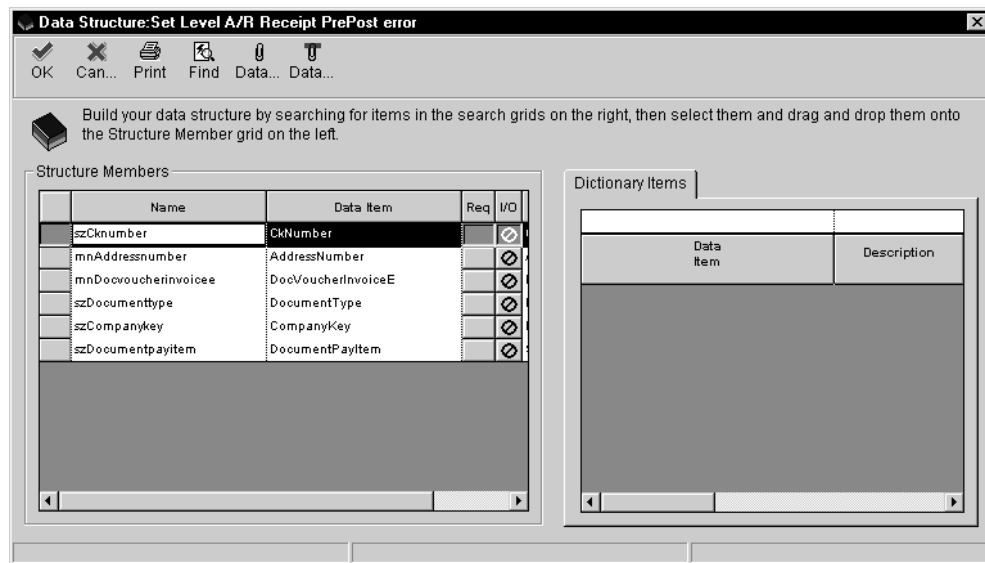
For example, DELM0025 would be the data structure created for the text substitution of level-break message LM0025. You may need to check the structure out to view individual fields.

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 some 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 Object Librarian. The name should use the same unique number that was used when creating the L xx data dictionary item. This number is to be appended to DEL xx . 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 into the business function you create. The typedef converts the data structure to C so it can be used in the business function.



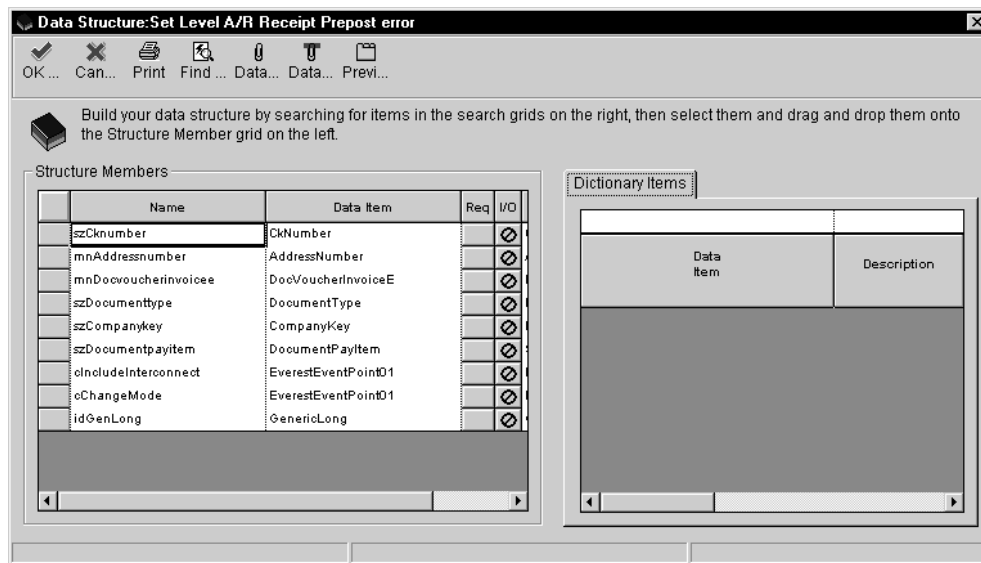
Refer to Data Structures for more information about data structures.

Creating a Level-Break Message Business Function Data Structure

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 J.D. Edwards-created business function data structures. J.D. Edwards recommends naming your own level-break messages using the following criteria: DLxxyy where xx refers to the system code you are using (between 50 and 55) and the yy is a unique number. This unique number should be the same number used previously. The maximum number of characters is eight.

For example, the J.D. Edwards-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. This data structure is not to be confused 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 OneWorld 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 OneWorldEventPoint01 to cIncludeInterconnect. This parameter is used as a flag to determine if the message is action. 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 a OneWorld application. This allows you to use level break messages in different applications, but not launch into application. 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 else other than as 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 Librarian for the new business function.

The name of the business function is the unique number preceded by BLxx. For example, if you named the level-break message data dictionary item L55025, you would name the business function BL55025. The J.D. Edwards naming convention is BLMxx.

2. Enter the name of the actual function on the Business Function Design form within Object Librarian.

When you name the function, the standard is to start the function with the name SetLevel_xx, where xx refers to the system code. Append the function with other descriptive words to identify what the level-break message is designed to do. For example, you might name the business function BLM0025 Set Level A/R Receipt Prepost Error.

3. After you name the business function, attach the business function data structure to it 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 a .h file.
5. With the Object Librarian function line complete (this includes any notes that you may want to attach), copy and paste the business function's typedef into its header file.
6. Create a typedef for the text substitution data structure DExx. To create the typedef, you will need to create a "dummy" function line.
7. After you create the 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 BLMxxxx.C business function.
9. Once you have copied the source into the new level-break business function, review the copied source and make the 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.

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.

In the "Variable declarations" section, you should have the following lines to declare the level-break message variables:

```
char  szForm[11];      /* Name of form in application */
char  szDDitem[11];    /* Data dictionary name of the level message */
char  szDLLName[11];   /* Name of the application DLL */
char  szDsTmp1[11];    /* Name of the text substitution data structure */
```

In the "Declare structures" section, you enter your own code for the appropriate type of level-break message. The following are examples from an existing business function:

```
DSDELM0002 dsTextData; /* Instance of text substitution structure */
FORMDSW0411Z1D dsFormData; /* Instance of form interconnect structure */
```

In the "Set pointers" section, you should have the following lines to ensure that the level-break message will work:

```
if (lpDS->idGenLong == (ID) 0)
{
    jdeSetGBRError (lpBhvrCom, lpVoid, (ID) 0, "4363");
    if (hUser)
    {
        JDB_FreeBhvr (hUser);
    }
    return ER_ERROR;
}
else
    lpDSwork = (LPDS_B0100011A) jdeRetrieveDataPtr (hUser, lpDS->idGenLong);
```

In the "Main Processing" section, you should have the following lines. Substitute your own items for the italicized items:

```
strncpy      ((char*) szDsTmp1, (const char*) ("DELM002"), sizeof (szDsTmp1) -1);
strncpy      (szDDitem, (const char*) ("LM0002"), sizeof (szDDitem));
memset       ((void*) (&dsTextData), (int) ('\0'), sizeof (dsTextData));
```

In the "Assign values from lpDS data structure to dsTextData here" section, you 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 are examples from an existing business function:


```
strncpy (dsTextData.szEdiuserid, (const char *) (lpDS-> szEdiuserid),
        sizeof(dsTextData.szEdiuserid));
strncpy (dsTextData.szEdibatchnumber, (const char *) (lpDS-> szEdibatchnumber),
        sizeof(dsTextData.szEdibatchnumber));
strncpy (dsTextData.szEditransactnumber,
        (const char *) (lpDS->szEditransactnumber),
        sizeof(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 is used for strings and the Mathcopy API for math numerics. If you use memcpy for dates, the characters are assigned directly.

In the “Assign values from lpDS data structure to dsTextData here” section, you should have the following lines to ensure that the level-break message will work:

```
JDBRS_GetDSTMPLSpecs (hUser, (char*) szDsTmpl, &lpDSwork->lpBlob->lpTDSMPL);
if (lpDSwork->lpBlob->lpTDSMPL != (LPDSTMPL) NULL)
{
    lpDSwork->lpBlob->lpTSTEXT=(LPSTR) AllocBuildStrFromDstmplName((LPDSTMPL)
        lpDSwork->lpBlob->lpTDSMPL, (char*) szDsTmpl,
        (LPVOID) &dsTextData);
    strncpy (lpDSwork->lpBlob->szDDItem, (const char *) (szDDItem),
        sizeof(lpDSwork->lpBlob->szDDItem));
}
if (lpDS->cIncludeInterconnect == '1')
```

In the “Form interconnect processing” section, you enter your own code for the appropriate level-break message. The following is an example from an existing business function. In the “Form interconnect processing” section, you should have the following lines to ensure that the level-break message will work.

```
strncpy (szDLLName, (const char *) ("P0411Z1"), sizeof (szDLLName));
memset ((void *) (&dsFormData), (int) ('\\0'), sizeof (dsFormData));
memset ((void *) (szForm), (int) ('\\0'), sizeof (szForm));
strncpy ((char *) szForm, (const char *) ("W0411Z1D"), sizeof (szForm) -1);
```

In the “Assign values from LpDS data structure to dsFormData” section, you enter your own code for the appropriate level-break message. The following is an example from an existing business function. This shows how you pass information from the data structure for the business function to the data structure for the form.

```
strncpy (dsFormData.EDUS, (const char *) (lpDS->szEdiuserid),
        sizeof(dsFormData.EDUS));
strncpy (dsFormData.EDBT, (const char *) (lpDS-> szEdibatchnumber),
        sizeof(dsFormData.EDBT));
strncpy (dsFormData.EDTN, (const char *) (lpDS->szEditransactnumber),
        sizeof(dsFormData.EDTN));
ParseNumericString(&dsFormData.EDLN, "1.0");
dsFormData.EV01 = '1';
```

In the “Get the form data structure id from the SVRDTL table” section, you should have the following lines to ensure that the level-break message will work:

```
lptam = jdeTAMInit (FILENAME_SVRDTL);
strncpy ((char *) lpDswork->lpBlob->szForm, (const char *) (szForm), size of
        (lpDswork->lpBlob->szForm) -1);
if (lptam != (LPTAM) NULL)
{
    lpASVRdtl = TAMAllocFetchByKey (lptam, INDEX4_ASVRDTL, szForm, 1);
    if (lpASVRdtl != (void *) NULL)
    {
        JDBRS_GetDSTMPLSpecs(hUser, (char*) lpASVRdtl->szFITEMplateName,
                            &lpDswork->lpBlob->lpFINDSMPL);
        if (lpDswork->lpBlob->lpFIDSMPL != (LPDSTMPL) Null)
        {
            lpDswork->lpBlob->lpFITEXT=(LPSTR) AllocBuildStrFromDstmplName
                ((LPSTMPL)
                 lpDswork->lpBlob->lpFIDSMPL,
                 (char*) lpASVRdtl->szFITEMplateName
                 (LPVOID) &dsFormData);
            strncpy (lpDswork->lpBlob->szDLLName, (const char *) (szDLLName),
                    sizeof(lpDswork->lpBlob->szDLLName));
        }
        TAMFree(lpASVRdtl);
    }
    TAMTerminate(lptam);
}
```

In the “Function Clean Up” section, you should have the following lines to 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 (source file B0100025.C – this is the actual name). 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 will receive all messages. If this field is filled in, it will supersede the following Address Number override parameter.	Leave blank or use any valid user ID. If left blank, all messages go to the UBE submitter.	Blank value
mnAddressnumber	User ID override. This field would be used if the user ID is not known but the address for that user is.	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 change the user who will be receiving the messages on the fly. 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 will be sent to the overridden mailbox. (Work Center use.)	Leave blank or use any valid mailbox designator. If left blank, all messages will be written to the default queue.	Blank value

idPPATworkField	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.

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 looking at the transaction record.

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

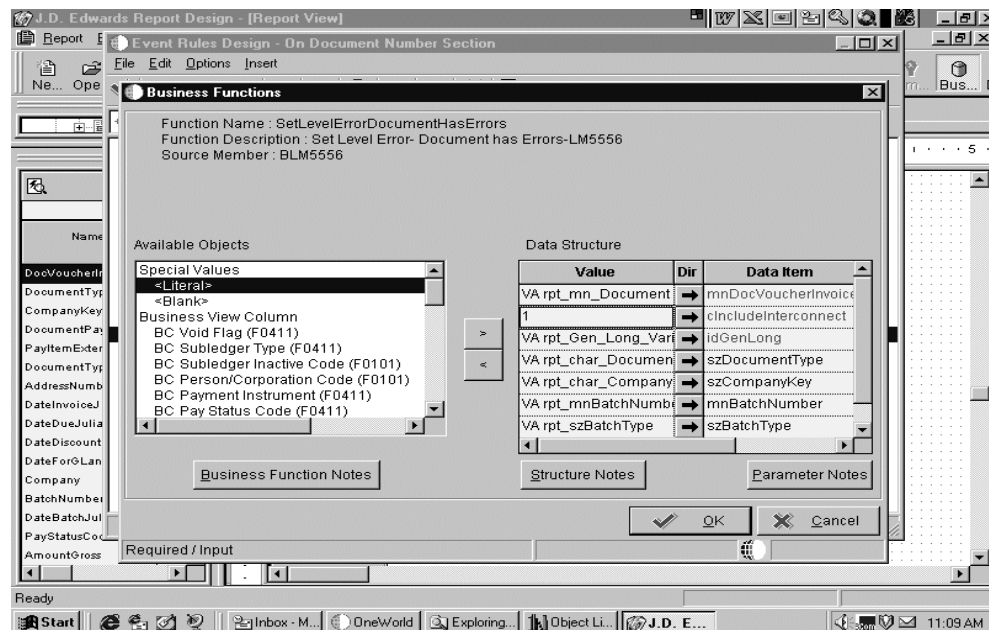
From Report Design

1. 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 edits for an individual record have taken place.

2. For each level-break established, do the following:
 - Call the level-break message business function at the appropriate level-break. The level-break message business function should relate to the type of error grouping that you want to occur at the particular level-break. For example, SetLevel_SFVoucher would group 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 you send the API will never be 1; it is only sent once, when you terminate the process.



- 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.

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 a level 2. The system automatically generates a level 1.

Parameter	Description	Allowed Values	Typical Values
mnLeveloftotaling	This parameter is the message level that you want the Work Center API to process.	Any valid number from 1–20. Note that the number “1” should only be used when the UBE is almost finished. A “1” will write the “Job Completed” message.	Numbers 1–20
idDataBaseWorkField	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
Parameter	Description	Allowed Values	Typical Values
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 left blank, errors that are in the “error space” are considered to be grouped by the level message being sent in the first parameter.	Blank value Default
szUserid	When the “cAllowUserIDToChange” flag has been set in the initialization function, this parameter will accept the new user ID.	Leave blank or use any valid user ID. If left blank, all messages are written to the last user ID used.	Blank value Default

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.

► To terminate the Work Center processes

When the batch program is about to terminate, call the Work Center error message function, `ProcessErrorsToPPAT`, one last time, sending it to level 1. The level 1 indicates the level of totalling is equal to 1 and it is completed. This will create the “job completed” message and will free any work space created by the Work Center API.

It is important that every report design using the Work Center API to process errors calls the API at the end of processing using a level 1. This should also be done by reporting jobs that are monitoring for any critical errors and need to terminate early.

When the report has finished processing, it creates Work Center messages, which can then be read using the OneWorld Work Center application. Batch errors are processed to JDEM system. Messages created are sent to the user who ran the report unless you specify that the message be sent to another user or administrator.

If no errors are encountered, the API sends the message that the job completed successfully to the Work Center.

Debugging

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 has been stopped, you can step through the code line by line to check such issues as flow of execution and data integrity.

This section describes the following:

- ☐ Working with the Event Rules Debugger
- ☐ Debugging Business Functions Using Visual C++
- ☐ Working with Visual C++ Debugger
- ☐ Debugging Strategies

In OneWorld, there are two debugging tools you can use:

- OneWorld Event Rules Debugger
- Microsoft Visual C++ Debugger

You use the Event Rules Debugger to debug event rules and the Visual C++ Debugger to debug C business functions. You can use the Event Rules Debugger to debug:

- Interactive applications
- Reports
- Table Conversions

Overview of the Debugging Process

The debugging process consists of determining where problems occur and fixing those problems. You should isolate each problem to a particular area, and then examine exactly how the program operates in that area.

If you make a change in your program while you are debugging with the Event Rules Debugger, you must:



- Stop the Event Rules Debugger
- Rebuild debug information
- Reset breakpoints
- Run the application

Following are some features available in the debuggers you can use:

Go	Visual C++ and Event Rules Debugger – Restarts program after a breakpoint has been reached. Visual C++ – Once you start an application, it will run until the breakpoint is reached. Event Rules Debugger – The application must initially be started from OneWorld Explorer.
Breakpoints	Visual C++ and Event Rules Debugger – Breakpoints 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.
Step	Visual C++ and Event Rules Debugger – The Step command will execute the current line of code. It 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 that line is executed.
Step Into	Visual C++ and Event Rules Debugger – This is 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. In OneWorld, the Step Into command can be used to debug a second application that is called from within a OneWorld calling application.
Skip	Visual C++ and Event Rules Debugger – Skip is used to skip execution of a line of code. The line of code which is not executed will turn red.
Step Over	Executes a single line of code. If the line of code is a function call, report interconnection, or form interconnection then that call will not be stepped into.

Stop

This stops the debugging process.

If you stop the Event Rules Debugger, the application continues to run, as if the debugger had not been started.

In Visual C++ the debugging process stops and the running application is also terminated.

Watch

Visual C++ and Event Rules Debugger – This lets you display the value of variables while the program is running. It also lets you inspect expressions, so you can see how a particular expression changes when variables change.

Interpretive and Compiled Code

OneWorld 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 lies within the application. Event rules scripting code used within Form Design and Report Design is interpretive. Interpretive code allows you to customize without going through a compile process every time you change something.

Compiled code is compiled and stored in an object file that may be called independently. The translation from program instruction to machine instruction happens at compile time. For example, table event rules, named event rules, and business functions are compiled. They are outside the application. They are also less subject to change. 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. Interpretive code is more flexible.

Working with the Event Rules Debugger

The Event Rules Debugger provides the essential debugging tools (breakpoints, step commands, and variable inspection), you need to debug a OneWorld application, report, or table conversion.

There are three main steps to set up and use the Event Rules Debugger:

- Building the debug information
- Running the Event Rules Debugger and setting breakpoints
- Running and debugging the application

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.

Working with Event Rules Debugger includes:

- ☐ Enabling debug information
- ☐ Understanding the Event Rules Debugger
- ☐ Debugging an application

Enabling Debug Information

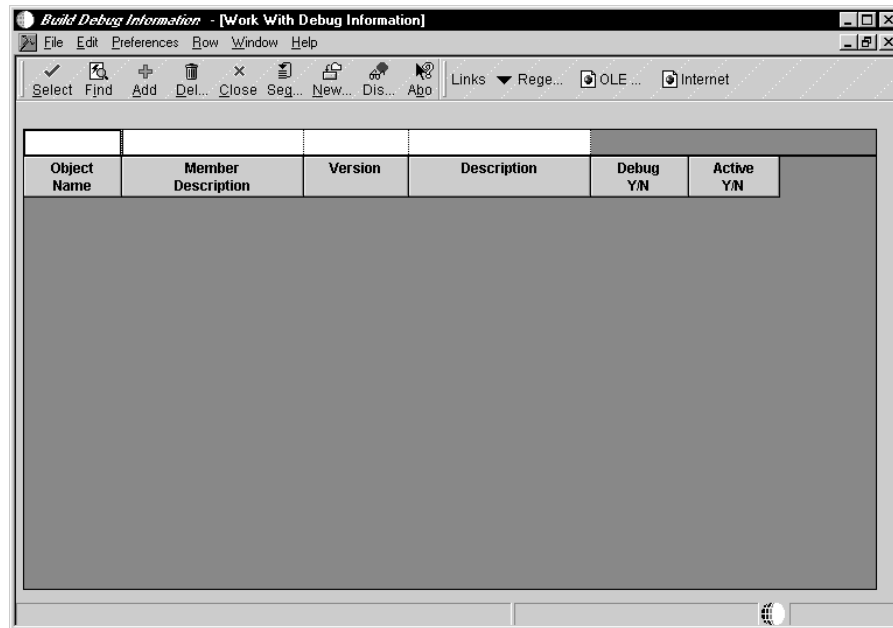
The Work With Debug Information form is the entry point to the Event Rules Debugger. Before you can debug an application, you must first prepare the application for debugging using the Build Debug Information application. This application creates a .DIA file for the application you want to debug. A .DIA file is a log file that contains the underlying operating instructions for the application.



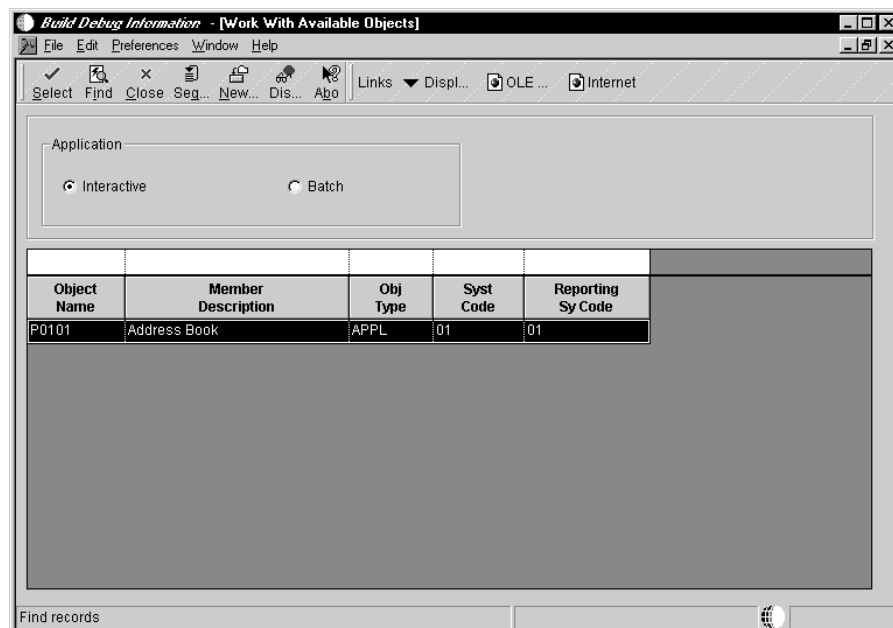
To enable debug information

From the Cross Application Development Tools menu (GH902)

1. Choose Build Debug Information.



2. On Work With Debug Information, choose the application you want to debug. If the application is displayed, choose Regenerate from the Row menu to build debug information. If the application is not displayed on Work with Debug Information, click the Add button to display the Work with Available Objects form.



3. On Work with Available Objects, search on either interactive or batch applications.
4. Choose an object to debug and click Select to build the debug information.

If you make any changes to event rules you must regenerate the debug information. This rebuilds the debug information file to include your revisions. The Event Rules Debugger must not be open when the debug information is regenerated.

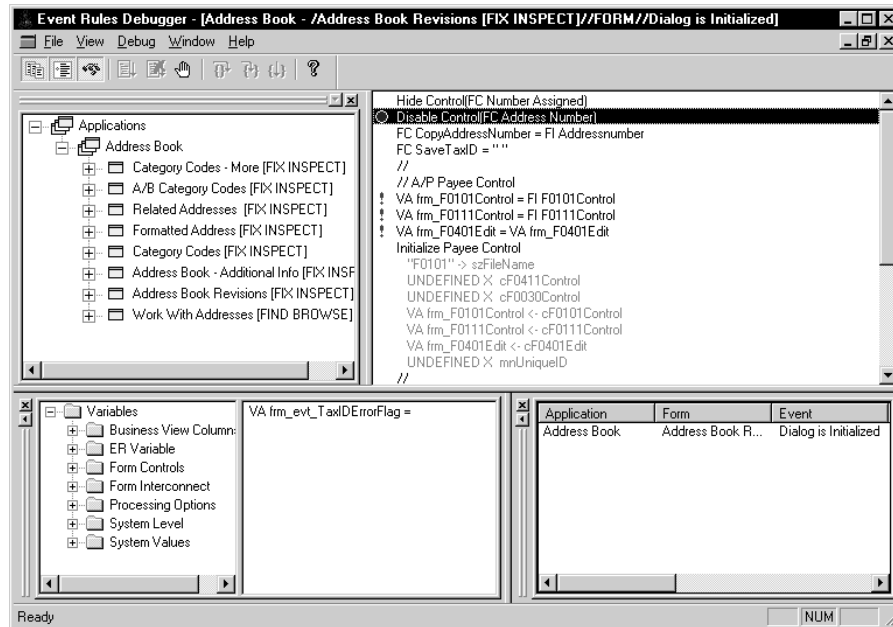
Field	Explanation
Object Name	The OneWorld architecture is object based. This means that discrete software objects are the building blocks for all applications, and that developers can reuse the objects in multiple applications. Each object is stored in the Object Librarian. Examples of OneWorld objects include: • Batch Applications • Interactive Applications • Business Views • Business Functions • Business Functions Data Structures • Event Rules • Media Object Data Structures
Member Description	The description of a record in the Software Versions Repository file. The member description is consistent with the base member description.
Version	A version is a user-defined set of specifications. These specifications control how applications and reports run. You use versions to group and save a set of user-defined processing option values and data selection and sequencing options. Interactive versions are associated with applications (usually as a menu selection). Batch versions are associated with batch jobs or reports. To run a batch process, you must choose a version.
Description	A user defined name or remark.
Debug Y/N	This field indicates whether debug information currently exists for the OneWorld application.
Active Y/N	This field indicates whether debug information for the OneWorld application is active.

Understanding the Event Rules Debugger

The Event Rules Debugger is a standalone tools program that consists of five main controls:

- Applications Tree control
- Event Rules windows
- Breakpoint Manager
- Variable Selection and Display control
- Search combo box

All controls except the event rule windows are dockable to any side of the main application. You can right-click on a control to dock a control or to hide it. If you close the Debugger, it will retain your docking settings so they are the same the next time you run the Debugger.



Application Tree Control

The Application Tree control lists applications that have debug information built and that are active in Build Debug Information (P98615). You can drill down through the tree to a specific event and open an Event Rules window for that event. The Application Tree control can list a maximum of ten applications at one time.

Event Rules Window

The Event Rules window displays the event rules for one event. The event name and path are displayed in the title bar for each Event Rule window. You can right-click to toggle the title between its long and short versions. The Event Rules windows show which line in the event rules is currently being executed.

The left hand side of an Event Rule window displays icons to describe the state of a line in the event rules, including the following states:

- 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 to do this:

- Double-click on the line in event rules
- Choose a line, and from the Debug menu choose Breakpoints
- Right-click on a line and choose breakpoints from the menu that appears

Variable Selection and Display Control

The Variable Selection and Display control consists of two controls separated by a splitter window. The left control is a tree control that lists the variable types as parent nodes and the current variables of that type as child nodes. The variables displayed are relative to the current Event Rule window. The right control is the variable display window. This window displays user selected variables and their current values. You can add a variable to the Variable Display window by double-clicking the desired variable in the Variables Tree.

You can change the value of variables while you are debugging an application. To change the value of a variable double-click on the variable in the Variable Display window. Change the value of the variable. The new value displays in the Variable Display window. If you enter an inappropriate value, for example if you change a numeric value to an alpha value, then the new value is not set and the value is not changed.

There are three special values that are displayed for variables:

blank	The value for the variable contains only blanks. This applies to string and character types only.
null	The value for the variable is set to ASCII “\0”.
unknown	The value for the variable could not be obtained from the engine. This happens when the applications are not running or the variables are out of scope.

Breakpoint Manager

Breakpoints tell the debugger where or when to break execution of a program. When the program is halted at a breakpoint, you can examine the state of your runtime structures, step through your event rules, and evaluate expressions using the Variable Watch window.

The Breakpoint Manager tracks which breakpoints are set and where they are located in an application. When you set a new breakpoint, an entry is made in the Breakpoint Manager. That entry contains the application name, form name, event name, and event rule line.

Right-click in Breakpoint Manager to perform the following operations:

- Delete breakpoints
- Delete all breakpoints
- Go to a breakpoint

You can also double-click on an entry in Breakpoint Manager to open the Event Rule window where 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 you wish to search for in the Search combo box and either press Enter F3. If the search text is found in your event rules text, the text is highlighted. If you press Enter or F3 again the next occurrence of your text will be highlighted.

The search control allows for 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”.

Following are the special characters you can use to perform more advanced searches.

^	The caret (^) indicates the beginning of the string. For example, the expression “^” 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 “[^0–9]” indicates that the target character should not be a d.
\$	The dollar sign (\$) matches the end of the string. For example, the expression “abc\$” 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 “a b” 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 there should be at least one match of the character to the left of the + sign in the expression.

- ?

The question mark (?) matches the character to its left 0 or 1 times.
- ()

The parenthesis affects the order of pattern evaluation and also serves as a tagged expression that you can use to replace a matched substring with another expression.
- []

Brackets ([and]) enclosing a set of characters indicate that any of the enclosed characters may match the target character.

Debugging an Application

Before You Begin

- ☐

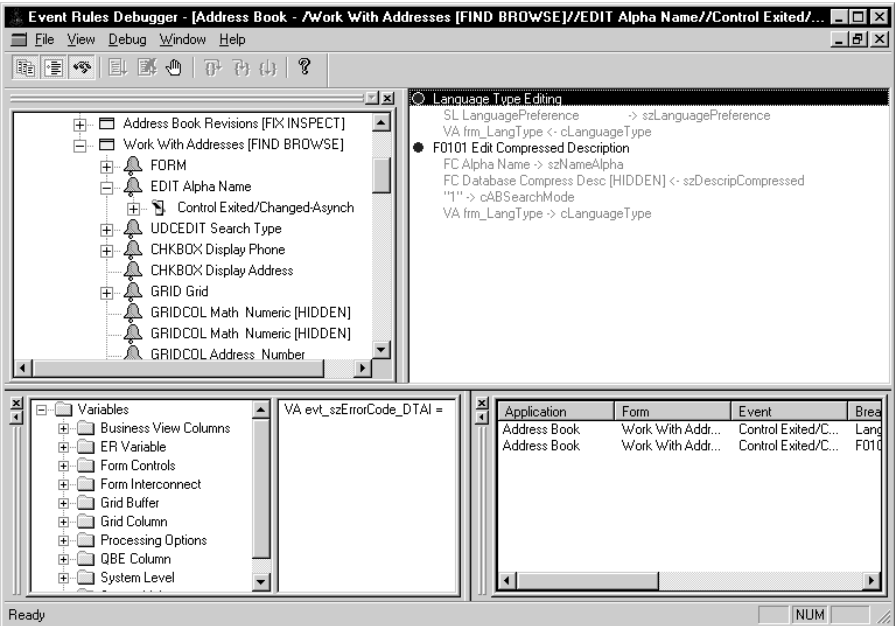
Before you can debug an interactive or batch application, you must enable the debug information table from Work With Debug Information. See *Enabling Debug Information*.

▶

To debug an application

From the Cross Application Development Tools menu (GH902)

1. Open Debug Application.



2. Choose a form (for interactive applications) or section (for batch applications) and an event to view.
3. Choose the event rule line where 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 OneWorld Explorer or from the Object Librarian, run the application you selected to debug.

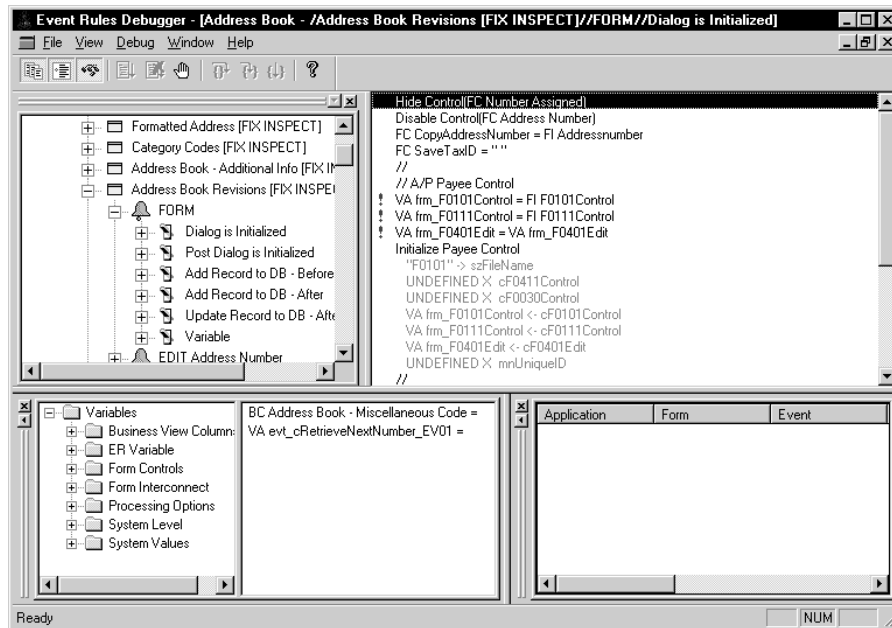
As your application encounters a breakpoint, the application pauses and displays the Event Rules Debugger.

While execution is stopped 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
 - Stop
 - Step Over
 - Step Into

Inspecting or Modifying a Variable

As you debug an application and encounter a point at which the interactive or batch application fails, modify the appropriate variable to correct it. Save your modifications and rerun the application to see if further debugging is required.



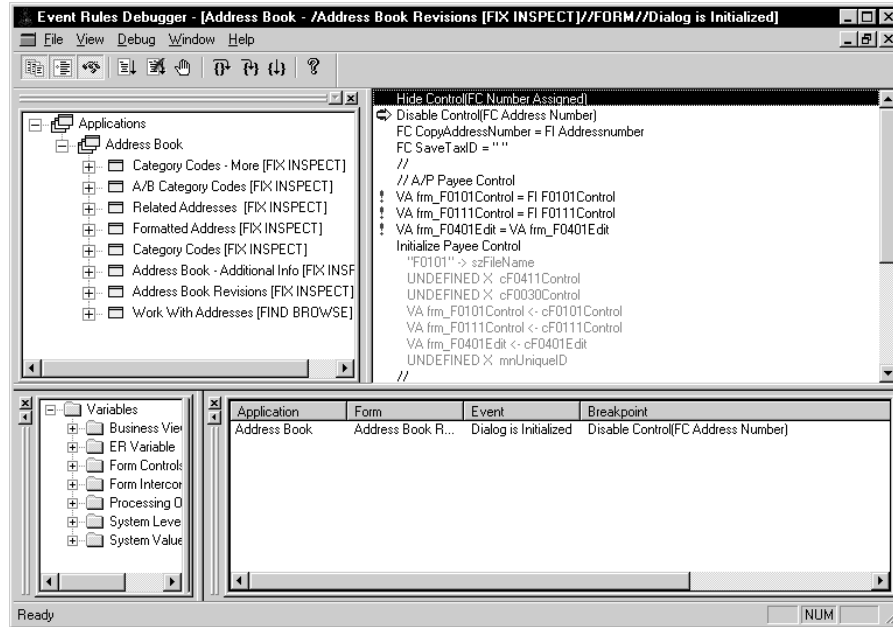
► To modify a variable

On event Rules Debugger

1. Double-click on a variable in the Event Rules variables window.
2. Revise information in the following field:
 - New Value

Just in Time Debugging

The Event Rules Debugger also uses “Just in Time Debugging.” Just in Time Debugging occurs after you build the debug information, run the Event Rules Debugger, and then run the application.



If the runtime engine has a problem resolving an event rule object or calling a business function, a message box appears that allows you to activate the Debugger. If you choose to activate the Debugger, then the Debugger is brought to the top and the event rule line that could not be processed is displayed with a yellow arrow to the left of it.

Setting Breakpoints

You should set at least one breakpoint on a starting event, such as *Dialog is Initialized*. Any event that you want the Debugger to stop on must have at least one line of event rule code. Do not use the Set Focus or Kill Focus events for debugging because these events typically happen so often that a loop may occur while you are debugging.

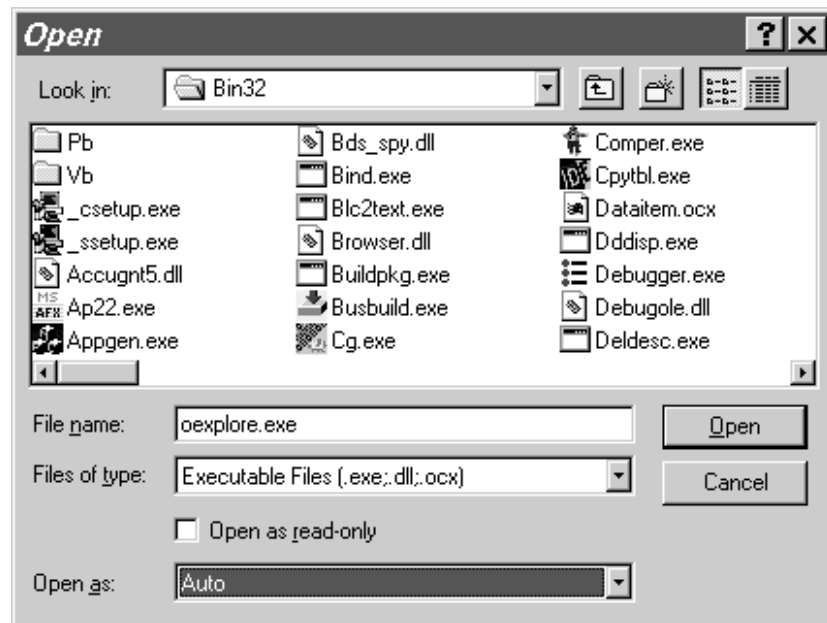
Debugging Business Functions Using Microsoft Visual C++

You can use Microsoft Visual C++ to debug business functions that are written in C. Starting with OneWorld B7331, you must use version six of Microsoft Visual C++. You can debug business functions attached to interactive applications or to batch applications.

Debugging Business Functions Attached to Interactive Applications

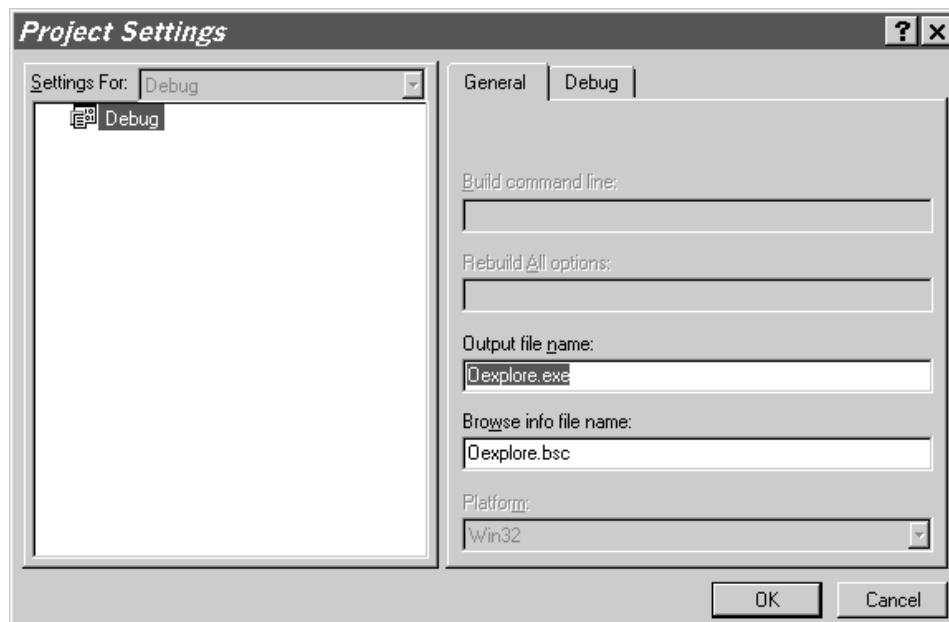
► To debug a business function attached to an interactive application

1. Make sure that OneWorld is not running. It must be closed to debug in this manner.
2. Open Visual C++ and make sure 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 your OEXPLORE.EXE on path \b7\System\bin32 and click OK.

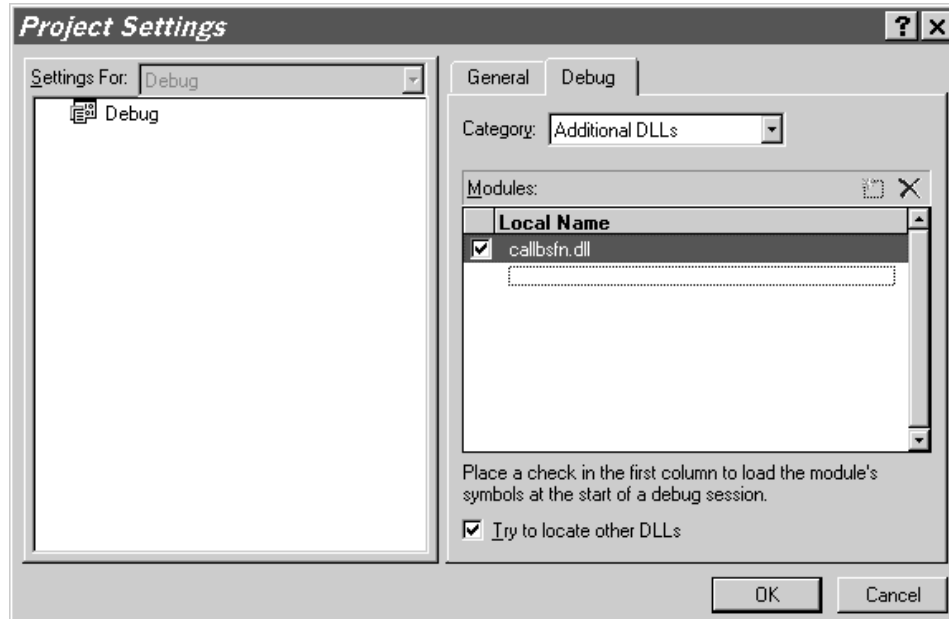


This creates a project workspace.

6. Choose your OEXPLORE.EXE project heading.
7. From the Project menu, choose Settings.

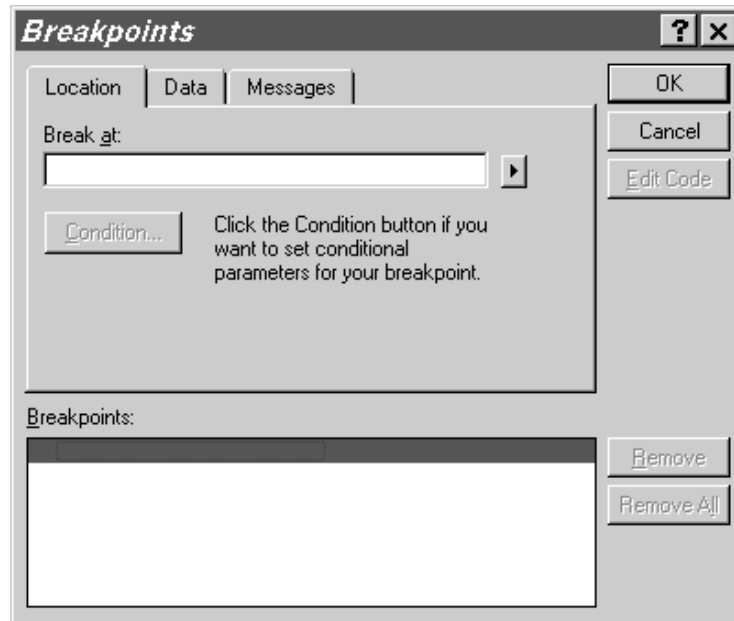


8. Click the Debug Tab.



9. In the Category list, choose Additional DLLs.
10. Click the Browse button to select the CALLBSFN.dll or other appropriate DLL on path \b7\bin32. (Note that this path may be different depending on your path code.)

11. Click OK.
12. Choose your .h and .c files for the source you want to debug from file open.
13. From the Edit menu choose Breakpoints to set breakpoints in your code.



If a “cannot open *.pdb” message appears, click through it.

If a message appears notifying you that breakpoints have been moved onto the next valid lines, it may indicate a source code and object mismatch. You may need to rebuild your business function.

14. From the Build menu, choose Start Debug, Go.

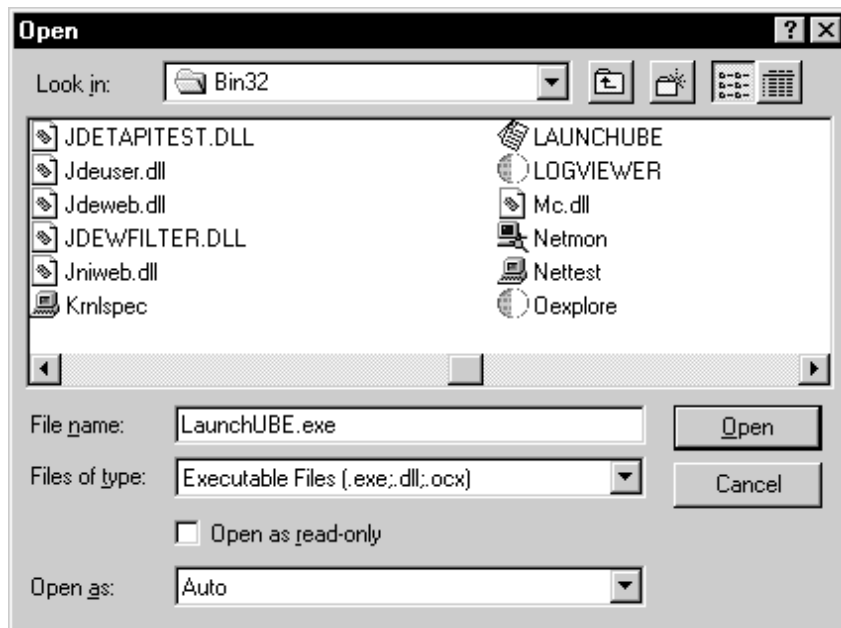
This causes the OneWorld signon box to appear.
15. Log into OneWorld as you normally would.
16. Execute the application. When it reaches the business function in Debug, it will open or display the C code in Visual C so that you can step through it.

If you are debugging business function event rules and C business functions, you can use the OneWorld debugger and the Visual C++ debugger together. Follow the steps above until you log into OneWorld. At this point follow the steps for the OneWorld debugger. Program execution stops if C code is accessed. You can then use Visual C++ to continue debugging. This is useful if you are trying to locate a problem and you are not sure whether the problem is in a C business function or the application calling the business function.

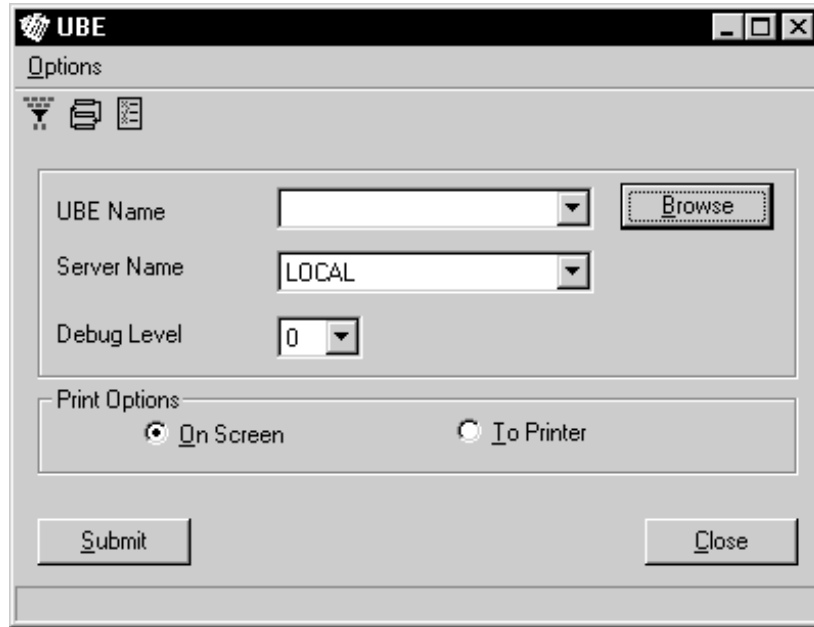
Debugging Business Functions Attached to Batch Applications

► To debug a business function attached to a batch application

1. Make sure that OneWorld is not running. It must be closed to debug in this manner.
2. Open Visual C++ and make sure 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 LaunchUBE.exe



6. On LaunchUBE browse for the report and version you wish to use.
7. Complete the Server Name field by selecting where the report will run.

You can choose Local or specify a server.

8. Set the Debug Level
9. Choose 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 C code. You can run the entire OneWorld system through the Visual C++ debugger (that is, you can start the oexplore.exe 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 ensure 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 Go Command

You can run a program using the Go command from the Debug menu. The program will run 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 may 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 executes until that line of code.

You set a breakpoint by placing the cursor anywhere on the line of code to stop before. Select Breakpoints from the Debug menu. A red octagon, similar to a stop sign, 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, double-click on the item 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 when debugging infinite loops.

Example: Debugging with the Visual C++ Debugger

In the following example, a breakpoint, Watch, Locals, and the Step command have been used.



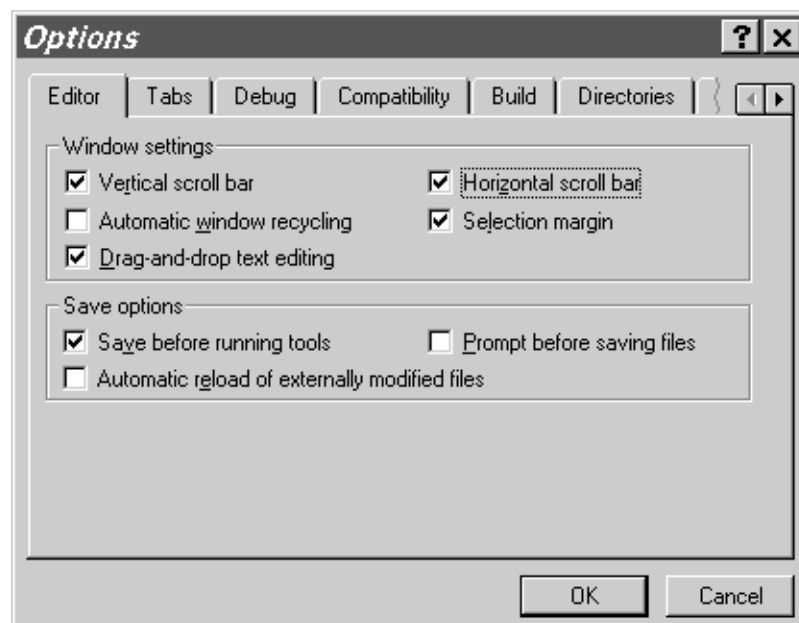
- A breakpoint has been set on the line of code with the call to `printf`. The red “stop sign” is to the left of the code. The call to `printf` has been executed with the Step command.
- The current line to execute has a yellow arrow to the left of it. It contains a call to the `getchar` function. That line of code has not yet been executed.
- `szGreeting` can be seen in the Locals window and in the Watch window. You can scroll these windows horizontally so you can see the entire line. You can see the beginning of the value of `szGreeting` in the Watch window in the example.
- 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. From the Tools menu, select Options.



2. On Options, customize your environment as desired.

Refer to the Visual C++ online helps for more information.

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?

The cause of 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 OneWorld-generated code, finding the problem may be more difficult. The debugger provides error messages that are of help. The most common problem has to do with missing objects. If there is a business function that is being called that does not exist, the tool will issue an error message that will identify 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, J.D. Edwards financial business functions use CFIN.DLL.

The object (BSN func) may 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 will be 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, it will immediately cause an unhandled exception in the program.

Termination of the program is more abrupt and less helpful when there are other kinds of objects missing. You must review all of the pieces of your application to make sure that they are all present and correctly built. A common error is to forget about media objects. If you cannot enter your program at all, a missing object is most likely the problem.

Make sure that the program is terminating in the same place. If the program is failing to free memory after its use, the program may eventually have insufficient memory to run. If so, you must reboot the workstation to free memory.

Is the Application Encountering Errors?

When a business function is called from event rules, its processing is unknown to the user. This 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.

There are two solutions to this sort of issue:

- 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?

If the output of the program is incorrect it means that there is a logic flaw in the code. Trace into the code to find the issue.

Where 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 you determine where the leak is, you can toggle between the grid rows in order 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 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 what you expected.

Set the break point at the beginning of the business function or at the beginning of an event like *Grid Row is Fetched*.

- Step through every code path.

Use the debugger to test the code. Both the J.D. Edwards 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 form 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 of the problem, not the symptom.
- Look for problems with object transfer/reinstall.

Debug Logs

There are several sections of the jde.ini file that 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. An application install of OneWorld 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 wish to debug must be rebuilt.

```
[JDE_CG]
STDLIBDIR=c:\MSDEV\LIB
TPLNAME=EXEFORM2
ERRNAME=CGERR
TARGET=Debug
INCLUDES=c:\MSDEV\INCLUDE;$(SYSTEM)\INCLUDE;$(SYSTEM)\INCLUDEV;
$(SYSTEM)\CG;$(APP)\INCLUDE;
LIBS=c:\MSDEV\LIB;$(SYSTEM)\LIB32;$(SYSTEM)\LIBV32;$(APP)\LIB32;
MAKEDIR=c:\MSDEV\BIN
USER=DEMO
```

You can output to 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. This is a very useful debugging tool when you have narrowed a problem down to a specific issue involving the JDEDB APIs.

```
[DEBUG]
Output=FILE
DebugFile=c:\jdedebug.log
JobFile=c:\jde.log
GlobalCompactSizeInit=1024
GlobalCompactSizeDestroy=0
```

The memory frequency setting allows you to validate the memory heap at a particular 1000th location. You can set breakpoints and go into the code.

```
[MEMORY DEBUG]
Frequency=10000
Full=0
```

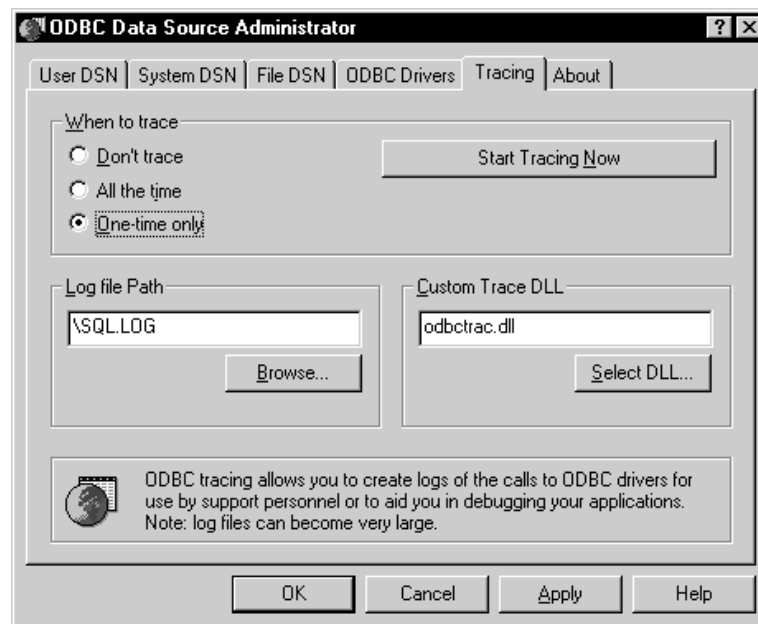
To view logs right click anywhere in the window and choose Log Viewer to view log files. For more information about other logs you can view for troubleshooting, refer to the appendices in the *Server and Workstation Administration Guide*.

SQL Log Tracing

You can use SQL Log Tracing to help you determine the exact SQL statement that is generated to the database.

► To turn on SQL Log tracing

1. From the Control Panel on your workstation, choose the 32 bit ODBC driver.
2. Click the tracing tab.
3. Choose one of the following radio buttons:
 - All the Time
 - One Time Only
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 output the results in a log file. This debug log is also useful for checking to see how the tool constructs a SQL statement.

Turning on Debug Tracing

 To turn on debug tracing

- 1. In the jde.ini under [DEBUG], change Output=NONE to one of the following values.

Output=FILE	This prints both database tracing and runtime tracing
Output=EXCFILE	This prints runtime tracing only.

- 2. Change Level= to suit specific debugging needs. Possible values for Level are contained in the comment line following the Level= line. Any combination is acceptable. Commas should separate values. The possible values for Level are:

EVENTS	This prints when an event is started and when it finishes.
BSFN	This prints when business functions are entered and when they return.
SF_?	This prints when system functions execute.

System Function Tracing

System functions are grouped by category (same categories as design time). Where reasonable, information has been provided about what 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 alias of the control and the control ID is printed.

The following example shows the jdedebug.log output running 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

```


Web Applications



Developing Web Applications

The OneWorld Development 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 the design of the forms. As you design a form, Form Design Aid allows you to view it in one of three modes:

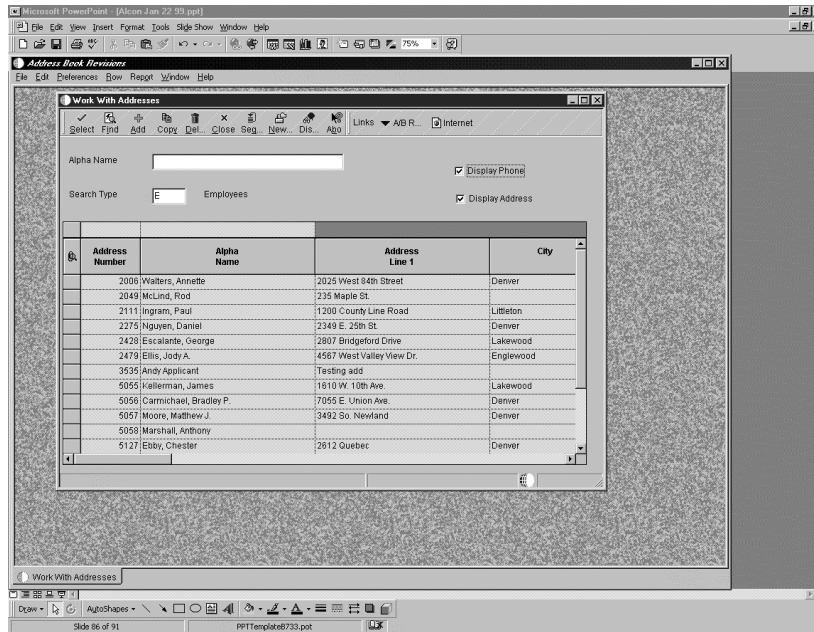
- Mode 1
- Mode 2
- Mode 3

The default mode for all applications is mode 1. You usually design your application in mode 1. You can create Windows, Java, or HTML applications.

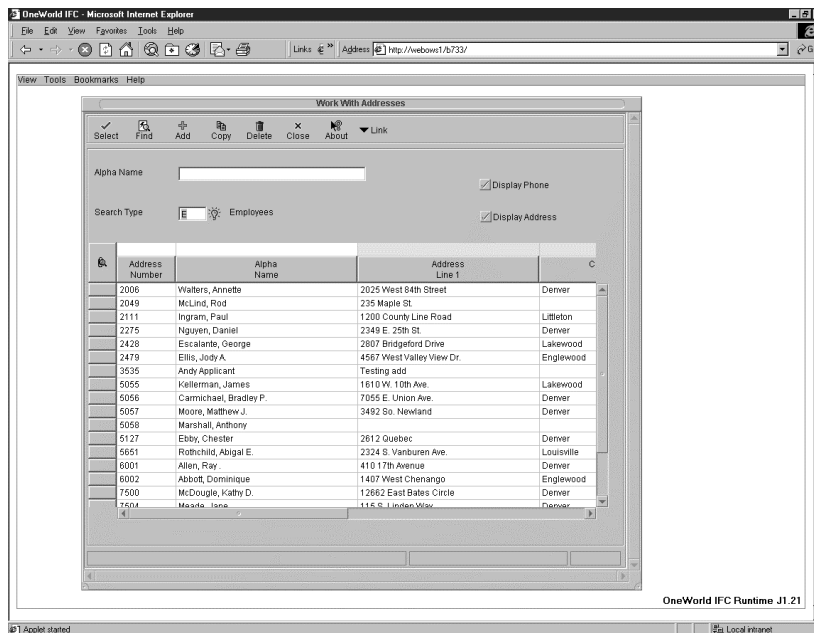
Most of the controls you put on a form are appropriate for any of the modes, but you can choose to show or hide (enable/disable) any control on the form for each of these three modes. In this way, you can develop one set of forms from which you generate three versions of the application (one for each mode). If you customize any of the OneWorld Web applications, or if you develop new ones, you must use the Web Generation Facility to generate them into Java or HTML applications. Refer to *Working with Controls* for more information about using different modes and customizing the controls on forms for different modes. You can use web versions to see different versions of a web application in different modes.

You typically create an application in Mode 1, the default mode. This application is automatically enabled for Windows. You can then use Mode 2 or Mode 3 to modify your Mode 1 application for Java or HTML.

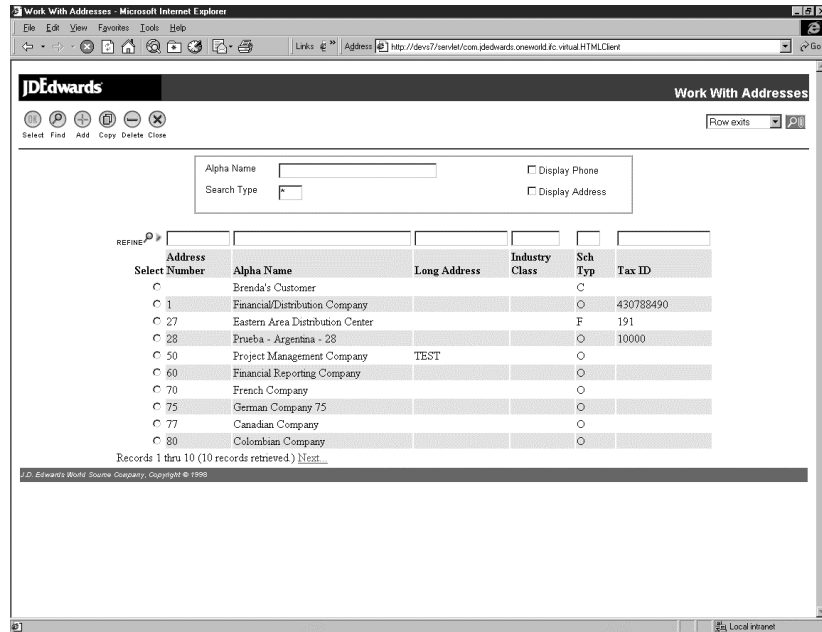




You can modify your Mode 1 application and use another mode, for example Mode 2, to create an application for Java.



You can use another mode, for example Mode 3, to further modify your Mode 1 application for an HTML application.



The three examples above illustrate some of the kinds of changes you might want to make in a windows application to make it easier for use on the Web. Typically you use fewer fields on a Web application. This makes it easier for an end user to view and use the application online.

Although you can develop or change Web-based applications (Java or HTML), on your workstation, you need a workstation that is web enabled to run the application to test it. All OneWorld clients are web enabled. The following components are required for your client:

- Internet Explorer 4.01 or above
- OneWorld Web Generation Facility (Installed with OneWorld client)
- Connectivity to a OneWorld Web server

If you customize any of the J.D. Edwards web applications you must regenerate them to create the specified Java or HTML applications. Or, if you create new applications, you must generate those to create Java or HTML applications.

After you create an application in form design you can use Menu Design to attach the application to a menu and designate which mode the application should run in. You should specify the mode on the menu so that the user can tell which application to use. Refer to *Menu Design* for more information about designing menus.

To test a Java application you must:

- Save your application in Form Design
- Generate the application to the Web server
- Attach the application to a menu and designate which mode it should run in

- Execute the application

Creating Web Applications

To create web applications, use the OneWorld toolset in the same manner as you would for standard applications.

If you use the OneWorld toolset to create your own applications, and these applications are designed for use as web applications, you should consider the following in order to ensure optimal performance:

- Each data and business function request executes 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 bottlenecks. 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 embedded event rules. This provides better performance than if you use C language business functions or event rules business functions (named event rules). Remember, whenever you call a business function the logic is performed on the remote OneWorld enterprise server. However, as mentioned above, if the event rules contain several table I/O or business function calls serially linked, these will perform better as a business function (named event rule) on the enterprise server.
- Design the web-ready application to get data in manageable chunks. If your application performs a business function request before and after each grid line, your performance will suffer. Also, you should try to avoid using business functions as a means to request data; use business views or table I/O instead.
- You should stress to users 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 only obtain 10 records at a time. If you develop custom applications for web use, you can consider defining query by example fields as required entries.

See Also

- *Understanding Forms Design*
- *Understanding Application Design*

-
- *Web Products Guide*

Generating Web Applications

The Web Generation Facility allows you to easily and intuitively generate a OneWorld application in either Java or HTML, or both. The look and feel of the tool is very similar to that of the other OneWorld development tools. Using the generator, you can generate by form or application.

Generating Web applications includes the following tasks:

- ☐ Generate a form
- ☐ Generate an application
- ☐ Generate all applications (administrative function)

After you generate a OneWorld form or application in Java or HTML, the tool stores the output in an appropriate database on the Web server. You select the database by specifying the data source. The output of the generation is a Java or HTML object, which is stored as persistent data in the database. The system retrieves the objects at runtime from the database. You can use the Web Generation tool to generate menus, data dictionary items, forms, tables, and business views.

Generate a form when:

- You change or add an existing Windows mode form
- You enable or disable Java or HTML modes on an existing form
- You create a new form

Generate an application when:

- You change or add existing Windows mode forms
- You enable or disable Java or HTML modes on existing forms
- You create new forms

Generate all applications when:

- You initially install OneWorld
- You upgrade OneWorld

The following table describes the generator options:

Generate by user	You choose whether to generate a public version or a personal version of the application. If you generate a public version, the system overrides the default version of that application with your updates allowing all users access to your changes. If you generate a personal version, only your logon is allowed access to the generated programs.
Status window	The Status window displays the generation process.
Stop the generation	After you start the generation, you can click Cancel to quickly stop the generation. If you stop the generation, only those forms already updated are changed. For example, if you stop an application generation during the fourth form of the application, only the first three forms are updated on the OneWorld Web server.
Delete objects from data sources	You can use the application generator to delete objects from the data sources. You specify the application and data source, and then choose the Delete option under the Action heading.
Generation messages	During the application generation, the following status messages appear: • Generation completed successfully. • Do you want to over-ride existing Default form? • Errors occurred during generation — see log file for more information.
Multiple language generation	Because OneWorld software supports multiple languages, the application generator also supports these languages using local workstation specification tables. However, before you can generate applications using multiple languages, you must have the desired translated specifications for the language loaded on the OneWorld developer client. See <i>Loading the Alternate Language Files</i> in the <i>Language Release Installation Guide</i> for information on loading multiple languages.

Generating a Form

If you changed or added an existing Windows mode application, if you added a new form, or if you enabled or disabled Java or HTML modes, you must generate the form before you can use the form. Use the Web Generation Facility to generate any changed or new forms.

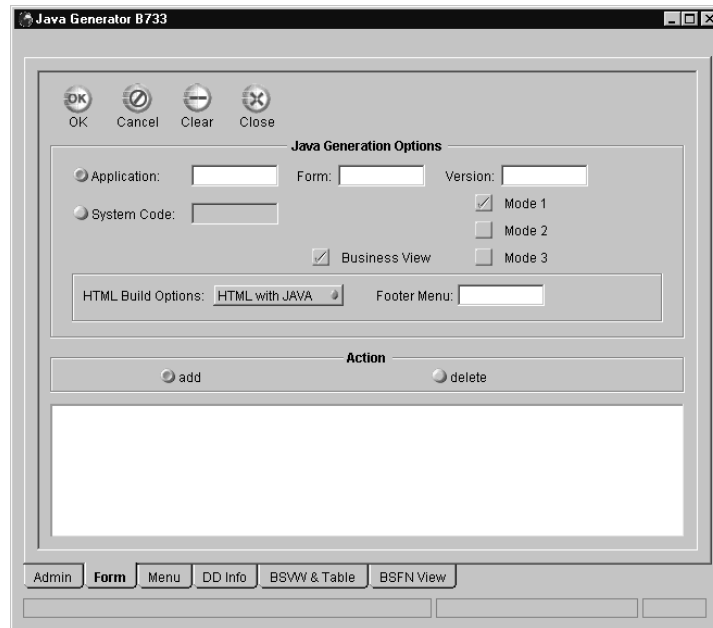
► To generate a form

From the Internet Tools menu (GH910), choose Web Generation Facility.

On Java Generator B733



1. Click the Admin tab and enter the Web server you are using in the following field:
 - Server Name
2. Choose a language from the available list.
3. Choose the appropriate log file options. You can specify whether to create a log, where to save the file, and what message types to create.
4. Click the Form tab and complete the following fields:
 - Application
 - Form
 - Version (optional)



5. Choose one of the following options under the HTML Build Options heading:
 - Java Only
 - HTML with Java
6. Choose one or more of the following modes:
 - Mode 1
 - Mode 2
 - Mode 3
7. Click the Business View option if you want to generate the associated business view and tables. If you do not choose this option, only the form is generated.
8. Click the following option from the Action heading and click OK:
 - Add
9. Review the status of the generation in the Status Window.

Field	Explanation
Server Name	The server on which to store the generated form or application.
Language	The language in which to generate the application.
Application	The application to be generated.
Form	The form to be generated. Use in conjunction with the application field.

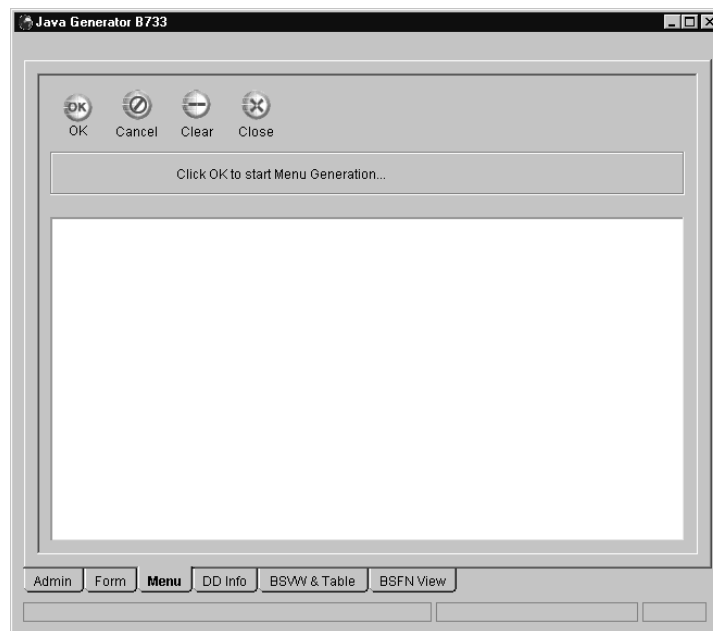
You can also use the Web Generation Facility to generate a:

- Menu
- Data Dictionary Information
- Business View and Table
- Business Function View

You can specify a menu to appear at the bottom of your form. This menu is known as a footer menu.

► **To generate a menu**

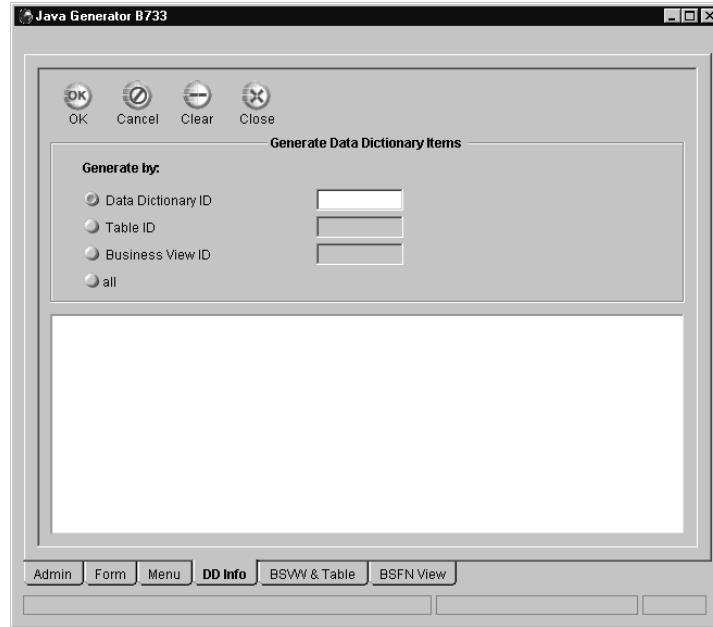
1. Click the Menu tab.



2. Click OK to start the menu generation.

► **To generate a data dictionary item**

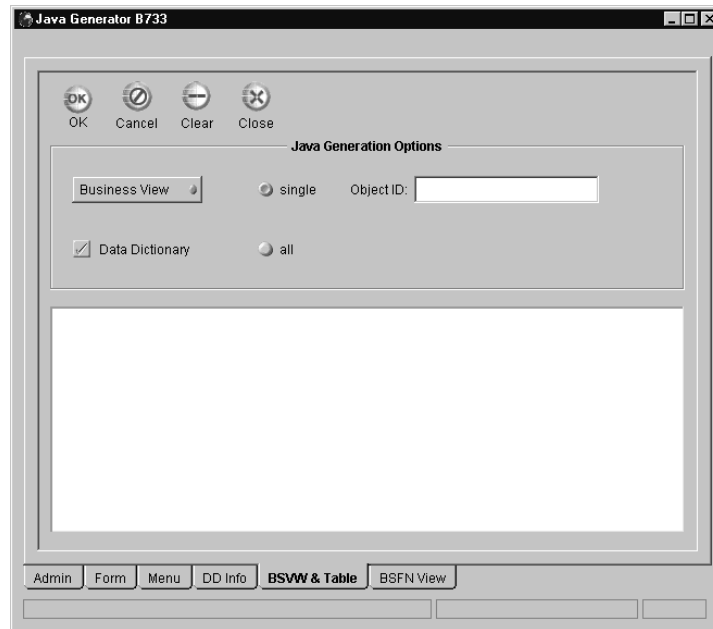
1. Click the DD Info tab.



2. Choose one of the following Generate by options:
 - Data Dictionary ID
 - Table ID
 - Business View ID
 - all

► **To generate a business view or table**

1. Click the BSVW & Table tab.



2. Choose one of the following options from the Business View menu
 - Business View
 - Table
3. Choose one of the following options for generation
 - single

If you choose single, complete the Object ID

- all
4. Click the Data Dictionary option to generate associated data dictionary items.

Generating an Application

If you changed one or more forms to an existing Windows mode application, if you created new forms to an existing Windows mode application, or if you enable or disable Java or HTML modes, you must generate the entire application before you can use any changed or new forms. Use the Web Generation Facility to generate an application.

To generate an application

From the Internet Tools menu (GH910), choose Web Generation Facility.

On Java Generator B733

1. Click the Admin tab and enter the Web server you are using in the following field:
 - Server Name
2. To specify the type of version to generate, choose one of the following options:
 - Public
 - Personal
3. Choose one of the following modes to generate:
 - Mode 1
 - Mode 2
 - Mode 3
4. Choose a language from the available list.
5. Choose the appropriate log file options. You can specify whether to create a log, where to save the file, and what message types to create.
6. Click the Form tab and complete the following field:
 - Application
7. Choose one of the following options under the HTML Build Options heading:
 - No HTML
 - To Server with Java
 - HTML with Java
8. Click the following option from the Action heading and click OK:
 - Add

9. Review the status of the generation in the Status Window.

Generating all Applications

Generate all applications only at initial OneWorld installation, when you have extensive changes or additions to all applications, or if you are installing an upgrade. This process takes several hours, is an administration function, and should not be done frequently.

To generate all applications

From the Internet Tools menu (GH910), choose Web Generation Facility.

On Java Generator B733

1. Click the Admin tab and enter the Web server you are using in the following field:
 - Server Name
2. To specify the type of version to generate, choose one of the following options:
 - Public
 - Personal
3. Choose one or more of the following modes to generate:
 - Mode 1
 - Mode 2
 - Mode 3
4. Choose a language from the available list.
5. Choose the appropriate log file options. You can specify whether to create a log, where to save the file, and what message types to create.
6. Click the Form tab and click the following option:
 - All
7. Choose one of the following options under the HTML Build Options heading:
 - HTML with Java
 - Java Only
8. Click the following option from the Action heading and click OK:
 - Add
9. Review the status of the generation in the Status Window.

Performance



Performance

This section includes tips and tools to help you enhance the performance of your applications.

This section includes information on the following topics:

- ☐ Performance Tips



Performance Tips

This chapter describes some things you can look for to improve performance. In particular it provides information about analyzing and improving the performance of:

- ☐ Data Dictionary
- ☐ Table Design
- ☐ Business Views
- ☐ Data Structures
- ☐ Data Selection and Sequencing
- ☐ Form Design
- ☐ Batch Applications
- ☐ Event Rules
- ☐ Business Functions
- ☐ Error Messaging
- ☐ Transaction Processing
- ☐ Troubleshooting

Things to Look For

Does the application seem slow on all platforms? If yes, then further analyze the application.

Does the application seem slow on a specific platform, but not all platforms? If yes, then there may be a logical issue or index issue for the given platform.

There are several tools to perform this analysis. Use `jddebug.log` first to determine at what event the application seems slow and what is happening on that event.

Analyze database and business function use. In analyzing the tables, consider the following:

- Records should be read in only once unless all secondary fetches for a given record were cached.
- The total of all JDB_OpenTable and JDB_OpenTableFromCache should match the total number of JDB_CloseTable calls for each table.
- The total number of actual JDB_OpenTable calls (the amount of opens that are not cached) should be as close to one as possible. Values larger than one indicate that multiple database cursors are being retained for the table.

Next turn on table logging for the jdedebug.log. Review the database I/O. If the table I/O is still a concern, start the SQL log and analyze the information in the log.

If you have justified all table I/O, and the application is still slow on certain events, then review the business functions running on those events. Make sure the business functions are running efficiently.

Look for business functions accessing a table a large number of times for the number of records being processed. Can caching be used?

Look for values that can be saved in global event rule variables instead of performing redundant I/O calls.

Look for multiple calls to multiple business functions running over the same table that can be consolidated into one table I/O call.

Look for business function or table I/O calls that can be moved to different events in order to reduce the number of times that they are called.

Look for business functions calling other business functions unnecessarily.

Data Dictionary Performance

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 OneWorld involves date and math fields. These fields require the most overhead for editing and formatting. Make sure 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 will

be increased, because disabling features causes some runtime overhead to be removed. Adding overrides such as edit/format triggers or next numbering will increase runtime overhead.

Validation

Validation is based on the data dictionary item that is attached to a control, column, or field. The overrides can also play a part in validation. Validation is the most costly for data items that have triggers. Validation of user defined codes also requires I/O to validate data values. General validation can be more costly for certain data types such as Math and Date. These data types require special logic to manipulate the OneWorld-specific data types.

Table Design Performance

Be careful using SELECT statements. Try to use existing indices for a table. It is better to use additional keys than to create a new index. Additional indices create overhead.

Use a partial key only for sequential and fetch next or just to look at a few more records.

If you open a table and then a fetch key, it destroys the pointer so do a fetch single instead. Almost any other database API will destroy the pointer also.

Opening a table the first time is a big performance hit.

The fetch matching key uses a greater than or equal key so it selects more than you may need. Use the correct JDB API for what you need.

Index Considerations

Adding a suitable index will almost always improve performance when selecting and fetching data from the database. However, each additional index adds maintenance overhead when records are added, updated or deleted in the database. The decision about adding a new database index over a table should balance these two factors.

OneWorld accesses 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 table is created but a warning message appears.

Inserting more than 1962 bytes into such a row or updating a row so that its total row size exceeds 1962 produces an error message and the statement fails.

Considerations for Coexisting with World Software

Make sure tables in OneWorld match AS/400 tables. If the data is not structured the same it causes problems.

If you plan custom modifications in a coexistence environment, the World RPG programs must be able to read the structure of any tables to be read by World Software. This means that the tables World Software must access should be initially created in World database structure, not from OneWorld using AS/400 foreign databases (Access, Oracle, SQL Server). The World environment must be set up first before OneWorld is set up on top of it.

Make date and time conversions on AS400 and OneWorld the same.

Refer to the *Coexistence Guide* for more information about coexistence.

Index Limitations for Various Databases

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 records 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 1 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.

Example:

In the F32944 table, the index is defined as:

```
{
  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 is not permitted, because the key length in this case is 1101 (>900).

Oracle

The following index limitations apply to Oracle:

- The maximum length for an index is 254 bytes, less the number of keys that allow NULL values.
- The maximum number of columns per index is 16 or a maximum key length of 900.

Key Column Violation

When you define indexes on various columns, ensure that no two indices, one of which may be a primary unique index, are defined on the same column.

Example:

In the F03B08 table, the indexes were defined as:

```
#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_F03B08, 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 is not permitted, because two indexes are defined on the same columns.

Specification File Corruption

If you encounter specification file corruption, recreate the tables in the source database.

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.

In order to leave 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 may 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, you should probably create several table I/O statements for specific requirements.

Business View Performance

Should I use a single-table or multiple-table business view?

In many instances, the application needs to 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 looking at the resulting 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, the second option is usually preferable because it makes best 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, it is likely that the same Company Master record will be retrieved for several records in the primary table, then it is usually preferable to fetch the company name explicitly using a business function or table I/O.

OneWorld 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 may need to choose between using an existing business view that may contain fields that are not needed or creating a new business view with only the desired fields. While any extraneous fields cause additional work on both the server and the workstation, this is usually *not* a significant performance concern. In cases involving tables that will be updated, OneWorld can automatically process all fields in a table, even if they are not included in the business view.

Minimizing the number of fields in a data structure is far more productive in improving runtime performance than minimizing the number of fields in a business view. You can create new business views that meet your specific needs that do not affect performance.

Unions

Unions are the most expensive database operation out of all the SQL SELECT statement types. In general you should avoid using them. An example of when you might want to use unions is if you are using the same part of tables at different times. For example, an application uses the same part of 10 different tables, and you only need that part one part at a time. Post Processes is an example of this.

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 just a side affect of another action.

Consider writing applications with the minimum number of grid columns required for the application's basic functionality. You can add columns to the associated business view to supplement the basic application with minimal effort. Therefore, it is better to add columns to a business view than to add columns to a grid.

Data Structure Performance

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 workstation and server.

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.

In order to leave space for literal values you should limit your structure size to 350 elements or less. Any data structure approaching a size of 100 elements or more should be considered carefully.

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 passing 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 data

structure values are checked. If they are not equal to blank, the Set User Selection system function is used to add these selections.

Form Design

This list contains standards for increased 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.
- Event rule variables should be used as work fields in place 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.
- Keep 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 linked list versus work files.

Find/Browse

This list contains standards for increased performance on Find/Browse forms.

- QBE assignments are not used (they negatively impact performance).
- The sort order on the grid should match both an index defined in OneWorld and a logical defined on the AS/400, 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. There may be additional fields 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

This list contains standards for increased performance on Header Detail and Headerless Detail forms.

- For maximum performance, the grid sort should match an index of the table in the business view. The index should exist on the AS400 and OneWorld tables.
- The sort order on the grid should match both an index defined in OneWorld and a logical file defined on the AS/400, 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. There may be additional fields 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, MMCU7 and the logical and index can include KIT, MMCU, TBM, BQTY.

Batch Application Performance

Setting Up Level Breaks

If the sort definition for the section is MMCU, KIT, BQTY, TBM and you want to have a level break on TBM, the level-break indicator must be set for all four fields. A level-break header or footer can then be attached or triggered off a change in TBM.

Slow Performance

If the server side performance for your batch process is extremely and you cannot determine the cause, you may want to check with your system administrator to ensure that the RequestedService setting in the JDE.INI file has the appropriate settings.

Event Rules Performance

When you are creating logic, if a row did not change then skip it. Don't 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 Fetch Keyed 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.

Use jdeCache instead of work tables

In early versions of OneWorld, developers used temporary local tables to contain working arrays. This technique is now obsolete. The jdeCache routines provide the 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, an EditGridLine function adds records to a cache and a corresponding EndDocument function retrieves the cached records and inserts them to the database. If the business functions are placed in different source members, a change in the CNC configuration may cause the application to fail.

Should I use JDB API calls in a business function or table I/O commands in event rule code to manage the database?

From a performance standpoint, there is little difference between the two methods. Make the choice based on the context. 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.

When possible, use JDBFetchNext instead of multiple JDBFetch statements

When multiple related database records must be read from the database, it is generally preferable from a performance standpoint to select the records at once and use JDBFetchNext to loop through the qualifying records. The alternative of making 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.

Make sure 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

A memory leak is when allocated memory is not freed when it is no longer needed. This can cause the performance of an application to degrade continuously as it 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.

One frequent source of unexpected memory leaks comes from failing to free allocated memory when processing errors or other conditions which may traverse an alternate execution path in a business function.

Look at 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

Be sure to match each jdbOpen with a corresponding jdbClose within the same business function for all possible execution paths. Serious performance problems can arise from unbalanced table opens and closes.

Error Messaging Performance

You can classify error messages in the data dictionary as errors or warnings. An error shows as a red stop sign and a warning is a yellow yield sign. Validation is automatically performed using the data dictionary. Other errors must be checked manually using event rules. Specify the field and error message number the error is from.

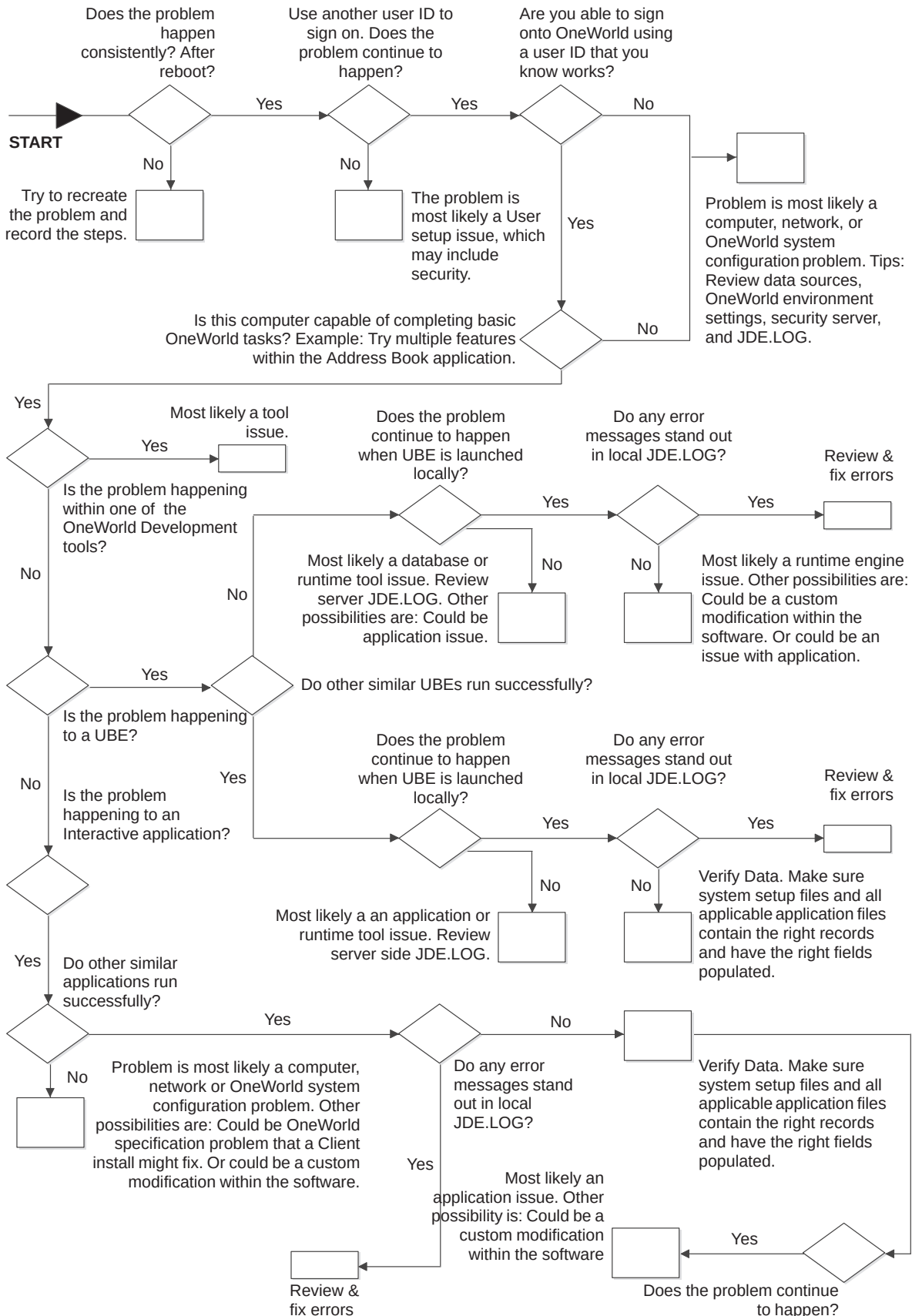
You must code an If statement in event rules so that if an error is already shown you do not show it again. If the error is only in the data dictionary, the error will show every time.

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 whole transaction rolls back.

Troubleshooting

The following diagram illustrates some steps you can take if you encounter problems developing your applications.



Appendices



Appendix A: OneWorld Modification Rules

Because the OneWorld Tools are comprehensive and flexible, you can customize certain aspects of business solutions and applications without making custom modifications. J.D. Edwards 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 options values
- Data dictionary attributes
- Workflow processes

This kind of flexibility improves efficiency and provides distinct advantages, such as the ability to do the following:

- Export grid records to other applications, such as a Microsoft Excel spreadsheet
- Resequence a grid on a different column
- Change grid fonts and colors
- Control major functionality using processing options.

However, even with the full commitment of J.D. Edwards to making the tools as flexible and robust as possible, if the need does arise to modify OneWorld, there are certain rules and standards 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. This should result in the least amount of disruption to your business and a reduction in the overhead cost involved during an upgrade because the upgrade time is reduced.

OneWorld keeps track of all custom modifications as you check them into the server. Using this feature, you can run the Object Librarian Modifications (R9840D) report prior to a merge to see a list of the changed objects.



OneWorld consists of control tables (such as menus, user defined codes, versions, and the data dictionary) and transaction tables (such as Address Book and the Sales Order File). J.D. Edwards ships the control tables with data that you can modify, while transaction files contain your business data.

Both of these kinds of tables go through an automatic merge process during an upgrade where control tables are merged with new J.D. Edwards data, while transaction files are converted to the new specifications, keeping your existing data. For the object specification merges (such as business view, tables, data structures, processing options, event rules, and applications), there are processes in place that merge the specifications or overlay them depending upon the rules defined under *What an Upgrade Preserves and Replaces*.

What an Upgrade Preserves and Replaces

If your business needs require custom modifications, observe the following general rules for OneWorld modifications to help ensure a smooth and predictable upgrade. These rules describe which of your modifications the upgrade process preserves and which modifications it replaces.

- “Preserve” means that during an upgrade, OneWorld automatically merges your modifications with the new J.D. Edwards applications shipped with the upgrade, and you do not lose your modifications. If there is a direct conflict between your specifications and J.D. Edwards’, the upgrade process uses yours. When there is no direct conflict between the two, then the upgrade process merges the two specifications.
- “Replace” means the upgrade does *not* merge those types of modifications and, therefore, J.D. Edwards replaces your modifications. You will need to do them again after the upgrade completes.

J.D. Edwards provides you with the Object Librarian Modifications (R9840D) report, which you can run before the upgrade process to identify objects that you modified.

For details about what OneWorld preserves and replaces during an upgrade, see the following:

- ☐ Understanding the general rules for modification
- ☐ Understanding interactive application rules
- ☐ Understanding report rules
- ☐ Understanding application text changes
- ☐ Understanding table specification rules
- ☐ Understanding control table rules
- ☐ Understanding business views rules
- ☐ Understanding rules for event rules
- ☐ Understanding data structure rules
- ☐ Understanding business function rules

Understanding Version Rules

Understanding Interactive Application Rules

Do not delete controls, grid columns, or hyperitems on existing OneWorld applications; instead, hide or disable them. OneWorld might use these for calculations or as variables, and deleting them might disable major functionality.

What an Upgrade Preserves

An upgrade preserves custom changes to the following:

- New applications. If you need to create an application, there are two ways to accomplish it. You can create an application from scratch, or you can copy an existing application. You can copy an application using the Application Design Aid's "Copy" feature to copy the application. This allows you to copy all of the application specifications, including event rules.

Use the Copy feature only to create a completely new application. If you use it to copy an existing application for some modifications, your new application *does not* receive any J.D. Edwards updates during an upgrade. For example, you could use Copy to make a copy of the Sales Order Entry (P4211) application, renaming it to P4211X. However, while the upgrade process replaces the Sales Order Entry (P4211) application, your P4211X application *does not* receive any changes that J.D. Edwards might have made to this application.

- New objects to existing J.D. Edwards forms. These include:
 - New hyperitems
 - New controls
 - New grid columns
- Any style changes, such as fonts and colors. New J.D. Edwards controls have the standard base definitions, so if you have adjusted the style, you need to do the same for any new controls added to an application
- Any code-generator overrides
- Data dictionary overrides
- Location and size changes for controls. If J.D. Edwards decides to place a new control in the new release of the software in the same place you have placed a custom control, the controls display on top of each other. This does not affect the event rules or the functionality of the application. After the upgrade, you can rearrange those controls using Application Design Aid.
- Sequence changes for tabs or columns. The upgrade process adds new J.D. Edwards controls to the end of your custom tab sequence. You can review the tab sequence after an upgrade.

What an Upgrade Replaces

An upgrade replaces custom forms on existing OneWorld applications. To create a custom form, you must create a custom application using system codes 55 - 59, and place the custom form on that custom application. From a performance standpoint there is no difference if an external application is called from a row exit or a form within the application. You can add custom form exits and row exits to J.D. Edwards applications; these custom exits can call your forms within your custom applications.

Understanding Report Rules

These rules apply to Report Design Aid specifications.

Do not delete objects on existing OneWorld reports; instead, hide them. OneWorld might use these for calculations or as variables, and deleting them might disable major functionality.

What an Upgrade Preserves

An upgrade preserves the following changes:

- New reports. If you create a report, there are two ways to accomplish it. One is to create a report from scratch, and the other is to copy an existing report. You can copy a report using Report Design Aid's "Copy" feature to copy the report. This allows you to copy all the report specifications, including event rules.

Use the Copy feature only to create a completely new report. If you use it to copy an existing report for some modifications, your new report *does not* receive any J.D. Edwards' updates during an upgrade. For example, you could use Save As to make a copy of the Print Pick Slips (R42520) report, renaming it to R42520X. However, while the upgrade process replaces the Print Pick Slips (R42520) report, your R42520X report *does not* receive any changes that J.D. Edwards may have made to this report.

- New objects to existing J.D. Edwards reports. These include:
 - Constants
 - Alpha variables
 - Numeric variables
 - Data variables
 - Run time variables
 - Database variables
 - Dictionary variables

- Any style changes, such as fonts and colors. New J.D. Edwards controls have the standard base definitions, so if you have adjusted the style, you need to do the same for any new controls added to a report
- Location and size changes for objects. If J.D. Edwards decides to place a new object, such as a control, in the new release of the software in the same place you have placed a custom object, the objects display right next to each other. This does not affect the event rules or the functionality of the report in any way. After the upgrade, rearrange those controls using Report Design Aid.
- Data dictionary overrides

What an Upgrade Replaces

An upgrade replaces custom sections on existing OneWorld reports. Use Report Interconnect instead and connect to a new custom report that uses system codes 55 - 59. There is no difference in the performance of a report being called through report interconnections.

Understanding Application Text Changes

An upgrade preserves overrides done in Application Design Aid, Report Design Aid, or Interactive or Batch Vocabulary Overrides.

Understanding Table Specification Rules

An upgrade merges your table specifications from one release level to the next.

What an Upgrade Preserves

An upgrade preserves the following changes:

- New tables
- Custom indices to J.D. Edwards tables

What an Upgrade Replaces

An upgrade replaces columns added or removed from existing J.D. Edwards tables. This includes changing field length, field type, and decimal position.

Instead of adding a new column to an existing J.D. Edwards table, use a tag file with system codes 55 - 59. For custom tag files, be aware of data item changes in the J.D. Edwards data dictionary. From one release to the next, J.D. Edwards 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 J.D. Edwards files, the upgrade process takes care of the data dictionary changes by upgrading the OneWorld database to the new release level. An upgrade preserves custom indices over the custom tag files.

Understanding Control Table Rules

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* J.D. Edwards tables, as the basis to do the data merge.

What an Upgrade Preserves

An upgrade preserves the following changes:

- Data dictionary custom changes, such as changes to row, column, and glossary text. The upgrade process uses your data dictionary as the base, and in case of a conflict with J.D. Edwards data items, your changes override. Create new data items using system codes 55 - 59.
- User defined codes. The upgrade process merges any new hard-coded J.D. Edwards values. (Values owned by J.D. Edwards are system 90 and above, and H90 and above.) The process also reports any J.D. Edwards hard-coded values that conflict with your custom values.
- Menus. In case of a conflict with J.D. Edwards base menus, your custom changes override.

What an Upgrade Replaces

An upgrade replaces columns added or removed from existing J.D. Edwards control tables.

Understanding Business Views Rules

Do not remove columns from existing business views. Changing business views that applications use can cause unpredictable results when you run the application. If you need to hide columns, do so at the application design level using either Application Design Aid or Report Design Aid. There is not much of a performance gain by deleting a few columns from a business view.

What an Upgrade Preserves

An upgrade preserves the following changes:

- New custom business views

- New columns, joins, or indices to the existing OneWorld business views

What an Upgrade Replaces

An upgrade replaces columns removed from J.D. Edwards business views.

Understanding Rules for Event Rules

An upgrade preserves the following changes:

- Custom event rules for the following custom objects:
 - Applications
 - Reports
 - Tables
 - Business Function event rules
- Custom event rules on a new custom control
- For J.D. Edwards applications, reports, tables, and business function events that *do not* have any J.D. Edwards event rules attached to the same event
- For J.D. Edwards applications, reports, tables, and business function events that have existing event rules attached, an upgrade disables and appends your custom event rules to the end of the event rules. The merge prints a report (R9840D) to notify you of any event rules that the upgrade process disabled.

To restore your custom event rules to J.D. Edwards objects, highlight and drag the event rules back to the proper place in the event and enable them. Prior to an upgrade perform the following:

- Run the Object Librarian Modifications (R9840D) report to identify modified applications
- Print the event rules for the modified application so that you can see the logic behind the event when restoring custom event rules

Understanding Data Structure Rules

What an Upgrade Preserves

An upgrade preserves all the following custom data structures:

- Forms
- Processing options

- Reports
- Business functions
- Generic text

What an Upgrade Replaces

An upgrade replaces the following custom modifications to J.D. Edwards data structures:

- Forms
- Processing options
- Reports
- Business functions
- Generic text

To bring your custom modifications to J.D. Edwards data structures forward to the next release level, run the Object Librarian Modifications (R9840D) report to list all of the modified data structures, and use this report as a guide for manually refitting data structure changes.

Understanding Business Functions Rules

For any new custom business functions, create a new (custom) parent DLL to store your custom modifications. See the *OneWorld Development Tools Guide* for details.

What an Upgrade Preserves

An upgrade preserves new custom business function objects.

What an Upgrade Replaces

An upgrade replaces modifications that you made to existing J.D. Edwards business functions and any business functions added to existing J.D. Edwards business function objects. To bring your custom changes forward to the next release level, run the Object Librarian Modifications (R9840D) report to list all of the modified business functions, and use this report as a guide for manually refitting the business function changes.

Always use the standard API (jdeCallObject) to call other business functions from within a business function. Not following this and all other standards will cause problems.

To determine modifications to the source of existing base J.D. Edwards business functions, use a third-party source-compare tool, such as Microsoft WinDiff. To determine modifications to APIs within business functions, see the OneWorld online help feature for the most current APIs.

Understanding Version Rules

For new custom versions, create a new version with a name that does not begin with XJDE or ZJDE. See the *OneWorld Foundation Guide* for details.

What an Upgrade Preserves

An upgrade preserves the following:

- Non-JDE versions
- Version specifications
- Processing option data

What an Upgrade Replaces

An upgrade replaces the following:

- All ZJDE and XJDE version specifications
- All processing option data for XJDE versions

In addition, processing option data is copied but not converted for non-JDE versions using JDE processing option templates. A warning is issued at runtime, and some data may be lost.

Also, event rule modifications for custom versions on JDE templates are not reconciled with the JDE parent template.



Appendix B: Form Processing

Processing for the following form types is discussed in this section:

- ☐ Find/Browse
- ☐ Parent/Child
- ☐ Fix/Inspect
- ☐ Header Detail
- ☐ Headerless Detail
- ☐ Search/Select
- ☐ Message Form

Processing for the following controls is also discussed in this section:

- ☐ Edit Control
- ☐ Grid Control



Process Flow for Find/Browse Form

Description

A Find/Browse form is used to query on a business view and select records from that business view for an operation. The following sections describe the processing flow of a Find/Browse form type.

Default Flags

There are no form option flags checked by default on Find/Browse forms. The only form option flag that can have an impact for this form type is the “No Fetch on Grid Business View” flag. Checking any of the other form option flags does not have any impact on the form processing.

The Find/Browse forms along with Parent/Child Browse forms are the only forms that have the “Entry Point” property checked by default. Find/Browse forms are typically the entry point into applications. The entry point property can be unchecked by the application developer.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Grid Fields
6. Initialize Static Text
7. Initialize Helps
8. Initialize Event Rules Structures
9. Create Toolbar
10. Load Form Interconnection data into corresponding business view columns and filter fields, if any
11. *Perform Event Rules: Dialog is Initialized*
12. *Perform Event Rules: Post Dialog is Initialized*
13. If the grid option “Automatically Find On Entry” is checked:
 - Begin Detail Data Selection and Sequencing

Header Data Retrieval

There is no header record for Find/Browse type forms.

Detail Data Selection and Sequencing

An internal structure is created representing the data selection and data sequencing requirements specified by the user. This is then passed to the database engine to do the actual database select and sequencing. The data is then held until the data is retrieved in the next step.

The data used for selection is pulled from filter fields and QBE Columns.

1. If the form option flag “No Fetch On Grid Business View” is unchecked:
 - Select and Sequence
 - Begin Data Retrieval
2. If the form option flag “No Fetch On Grid Business View” is checked no data retrieval is performed.

Data Retrieval

A request is issued to the JDEKRN, which performs the actual fetch of the data from the database. It will read one record at a time and for each record, perform the following processing:

1. Attempt to fetch a record from the database
2. If a record is fetched
 - Copy the data into the business view data structures
 - *Perform Event Rules: Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this grid record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Grid Line - Before*
 - Add the row to the grid. The row now exists in the grid control
 - Perform Event Rules: *Write Grid Line - After*
 - Clear the grid data structures for reading the next record
 - Remove the suppress grid line flag

The previous steps occur for each record read from the database. After all records have been read:

3. Perform Event Rules: *Last Grid Record Has Been Read*

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Thread Handling
5. Terminate Helps
6. Free all structures for form, including structures for Business View Columns, Form Controls, Grid Columns, and Event Rules
7. Destroy the window

Menu/Toolbar Items

Select

Select is a standard item that is automatically placed on Find/Browse forms. There is no default processing for Select on Find/Browse forms. It will behave as a user defined item.

Close

Close is a standard item that is automatically placed on Find/Browse forms. It closes the form.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*
3. Begin Closing Form

Find

Find is a standard item that is automatically placed on Find/Browse forms. When clicked by the user, it is the signal to the runtime engine to call the database and reload the grid based on the selections in the filter fields.

1. Perform Event Rules: *Button Clicked*
2. If there no errors in any filter fields:
 - Begin Data Selection and Sequencing
 - Perform Event Rules: *Post Button Clicked*

Delete

Delete is a standard item that can be added to Find/Browse forms. It deletes a record in the grid from the database.

1. Perform Event Rules: *Button Clicked*
2. For each selected grid row that is deleteable
 - Copy the grid row data into the Business View structures
 - Remove Suppress Delete flag
 - *Perform Event Rules: Delete Grid Rec Verify - Before*
 - If the Suppress Delete flag is not set
 - Display Delete Confirmation dialog. If the user clicks NO or CANCEL the rest of this processing is skipped.
 - *Perform Event Rules: Delete Grid Rec Verify - After*
 - If the Suppress Delete flag is not set

Perform Event Rules: Delete Grid Rec from DB - Before

If the Suppress Delete flag is not set:

- Delete the record in the Business View from the database
- Delete the grid row from the grid control
- Perform Event Rules: *Delete Grid Rec from DB - After*

3. Perform Event Rules: *All Grid Recs Deleted*
4. Perform Event Rules: *Post Button Clicked*

Copy

Copy is a standard item that can be added to Find/Browse forms. There is no default processing for copy on Find/Browse forms. They will behave as user defined items. If a form Interconnection is placed on the Button Clicked event then the called form will open in copy mode.

Add

Add is a standard item that can be added to Find/Browse forms. There is no default processing for add on Find/Browse forms. They will behave as user defined items. If a form Interconnection is placed on the *Button Clicked* event then the called form will open in add mode.

User Defined Items

User-defined items are nonstandard items you can add to Find/Browse forms to perform specialized processing not handled by the standard items.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*

Process Flow for Parent/Child Form

Description

A Parent/Child form is used to query on a business view and represent the data in a hierarchical manner. Records can also be selected from that business view for an operation. The following sections describe the processing flow of a Parent/Child form type.

Default Flags

There are no form options flags checked by default on Parent/Child forms. The only form option flag that can have an impact for this form type is the “No Fetch on Grid Business View” flag. Checking any of the other form option flags does not have any impact on the form processing.

The Parent/Child Browse forms along with Find/Browse forms are the only forms that have the “Entry Point” property checked by default. Parent/Child forms are typically the entry points into applications. The entry point property can be unchecked by the application developer.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Grid/Tree Fields
6. Initialize Static Text
7. Initialize Helps
8. Initialize Event Rules Structures
9. Create Toolbar
10. Load Form Interconnection data into corresponding business view columns and filter fields, if any
11. Perform Event Rules: *Dialog is Initialized*
12. Perform Event Rules: *Post Dialog is Initialized*
13. If the grid option flag “Automatically Find On Entry” is checked

- Begin [Detail Data Selection and Sequencing](#)

Header Data Retrieval

There is no header record for Parent/Child forms.

Detail Data Selection and Sequencing

An internal structure is now created representing the data selection and data sequencing requirements specified by the user. This is then passed to the database engine to do the actual database select and sequencing. The data is then held until the data is retrieved in the next step.

The data used for selection is pulled from filter fields and QBE Columns.

1. If the form option flag “No Fetch On Grid Business View” is unchecked
 - Select and Sequence
 - Begin [Data Retrieval](#)

Data Retrieval

A request is issued to the JDEKRN, which performs the actual fetch of the data from the database. It will read one record at a time and for each record, perform the required processing. The data retrieval is performed in two different ways in the Parent/Child browse form. The fetch performed to get the first level nodes in the tree is similar to the one performed in the Find/Browse form:

Data Retrieval for the first level node in the tree

1. Attempt to fetch a record from the database
2. If a record is fetched
 - Copy the data into the business view data structures
 - Perform Event Rules: *Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Grid Line - Before*
 - Add the row to the grid and the corresponding column in Tree. The row and the corresponding tree node now exist in the control.
 - Perform Event Rules: *Write Grid Line - After*
 - Clear the data structures for reading the next record.

The previous steps occur for each record read from the database. After all records have been read:

1. If the Parent/Child Browse form has the grid permanently hidden, then expand the header node and move the selection to the first child node under the header. This initiates the events: 'Tree - Node Level Changed' and 'Tree - Node Selection Changed.'
2. Perform Event Rules: *Last Grid Record Has Been Read*

The Parent/Child form also performs data retrieval whenever a particular node on the tree is expanded. However, this fetch is performed only once for each node that is expanded. If a particular node is collapsed and expanded again, then the tree and the grid are replenished from internal structures. The processing performed for node expand event is given below:

Data Retrieval for Tree Node Expand

1. Perform Event Rules: *Tree - Node is expanding*
2. Check for internal flags to see if the fetch is not suppressed. If the fetch is suppressed through the system function, *Suppress Fetch on Node Expand*, the processing stops here.
3. Perform the key substitution (copying child key into parent key), that was setup in the design of the form.
4. Attempt to fetch a record from the database
5. If a record is fetched
 - Copy the data into the business view data structures
 - Perform Event Rules: *Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Grid Line - Before*
 - Add the row to the grid and the corresponding column in Tree. The row and the corresponding tree node now exist in the control.
 - Perform Event Rules: *Write Grid Line - After*
 - Clear the data structures for reading the next record.
6. Perform Event Rules: *Last Grid Record Has Been Read*
7. Move the tree selection to the first child node under the expanding node. This initiates the Events: *Tree - Node Level Changed* and *Tree - Node Selection Changed*.

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Thread Handling
5. Terminate Helps
6. Free all structures for form, including structures for Business View Columns, Form Controls, Grid Columns, and Event Rules.
7. Destroy the window

Menu/Toolbar Items

Select

Select is a standard item that is automatically placed on Parent/Child Browse forms. There is no default processing for Select on Parent/Child Browse forms. It will behave as a user-defined item.

Close

Close is a standard item that is automatically placed on Parent/Child Browse forms. It closes the form.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*
3. Begin Closing Form

Delete

Delete is a standard item that can be added to Parent/Child Browse forms. Similar to the Find/Browse form, the Parent/Child Browse form does the following.

1. Perform Event Rules: *Button Clicked*
2. For the selected tree node
 - Copy the grid row data into the Business View structures
 - Remove Suppress Delete flag
 - Perform Event Rules: *Delete Grid Rec Verify - Before*
 - If the Suppress Delete flag is not set
 - Display Delete Confirmation dialog. If the user clicks No or Cancel the rest of this processing is skipped.

- Perform Event Rules: *Delete Grid Rec Verify - After*
- If the Suppress Delete flag is not set
- Perform Event Rules: *Delete Grid Rec from DB - Before*
- If the Suppress Delete flag is not set
 - Delete the record in the Business View from the database.

This will not delete the child records of the node, if any, from the database. It is your responsibility to delete them from the database.

- Delete the grid row from the grid control
 - Perform Event Rules: *Delete Grid Rec from DB - After*
3. Perform Event Rules: *All Grid Recs Deleted*
 4. Perform Event Rules: *Post Button Clicked*

Find

Find is a standard item that can be added to Parent/Child Browse forms. When clicked by the user, it is the signal to the runtime engine to call the database and reload the grid based on the selections in the Filter Fields and QBE columns.

1. Perform Event Rules: *Button Clicked*
2. If there no errors in any filter fields
 - Begin Data Selection and Sequencing
 - Perform Event Rules: *Post Button Clicked*

Copy

Copy is a standard item that can be added to Parent/Child Browse forms. There is no default processing for copy on Parent/Child Browse forms. They will behave as user defined items. If a form interconnection is placed on the Button Clicked event then the called form will open in copy mode.

Add

Add is a standard item that can be added to Parent/Child forms. There is no default processing for add on Parent/Child forms. They will behave as user defined items. If a form interconnection is placed on the *Button Clicked* event then the called form will open in add mode.

User Defined Items

User-defined items are nonstandard items that you can add to Parent/Child forms to perform specialized processing not handled by the standard items.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*

Process Flow for Fix/Inspect Form

Description

A Fix/Inspect form is used to update and insert single data base records. It is also referred to as a Single Record Maintenance form. This form type can also be used to display static information to the user as well as prompt the user for information, for example a Sign-on Screen. The following sections describe the processing flow of a Fix/Inspect form type.

Default Flags

None of the form option flags are set by default.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Static Text
6. Initialize Helps
7. Initialize Event Rules Structures
8. Create Toolbar
9. Load Form Interconnection data into corresponding business view columns.
(Note: Form controls and business view columns are sharing internal memory so copying into the business view is effectively copying into the internal form control memory locations.)
10. Perform Event Rules: *Dialog is Initialized*
11. If the “No Fetch Form Business View” is checked
 - Perform Event Rules: *Post Dialog is Initialized*
12. If the form is in Add Mode
 - Begin Clearing Dialog
13. If the form is in update mode and “No Fetch Form Business View” is checked
 - Change mode to add mode
 - Display the internal form control values to the screen

- Begin Clearing Dialog
- 14. If the form is in update mode and “No Fetch Form Business View” is not checked
 - Begin Data Retrieval

Data Retrieval

A request is issued to the JDEKRNL, which performs the actual fetch of the data from the database. The fetch will be based on the information passed to the form through its form data structure.

1. Attempt to fetch a record from the database
2. If a record is fetched
 - Copy the data into the business view data structures (Note: remember that copying into the business view is effectively copying into the internal form control memory locations.)
3. If a record is found and the form is in Copy Mode
 - Switch to Add Mode
 - Display the internal form control values to the screen
 - Begin Clearing Dialog
4. If a record is found and not in Copy mode.
 - Perform Event Rules: *Post Dialog is Initialized*
5. If no records were fetched
 - Switch to Add Mode.
 - Display the internal form control values to the screen
 - Begin Clearing Dialog
6. Set the default focus based on the resulting data base mode

Clearing Dialog

1. If the form was called in Copy Mode
 - Clear the key (primary index) controls that have the “Do Not Clear After Add” flag unchecked
2. If the form was not called in Copy Mode
 - Clear all form controls that have the “Do Not Clear After Add” flag unchecked
3. Perform Event Rules: *Clear Screen Before Add*
4. Perform Event Rules: *Post Dialog is Initialized*

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Thread Handling
5. Terminate Helps
6. Free all structures for form, including structures for Business View Columns, Form Controls, and Event Rules.
7. Destroy the window

Menu/Toolbar Items

OK

OK is a standard item that is automatically placed on Fix/Inspect forms. It validates the information on the form and updates or adds to the database through JDEKRNL function calls.

1. If there are any errors/warnings on the form, stop OK processing
2. *Perform Event Rules: Button Clicked*
3. For each control on the form
 - If the current control is a form control and it has not passed validation
 - *Perform Event Rules: Control is Exited*
 - *Perform Event Rules: Control is Exited and Changed - Inline*
 - *Perform Event Rules: Control is Exited and Changed - Async*
 - Perform Data Dictionary validation
4. If there are any errors on the form, stop OK processing
5. If the form is in Add mode
 - *Perform Event Rules: Add Record to Database - Before*
 - If the database add has not been suppressed
 - Add the record to the database
 - Perform Event Rules: Add Record to Database - After
6. If the form is in Update mode
 - *Perform Event Rules: Update Record to Database - Before*
 - If the database update has not been suppressed

- Update the record to the database
 - Perform Event Rules: Update Record to Database - After
7. Perform Event Rules: Post Button Clicked
 6. If form is in Add Mode
 - If form was called in Copy Mode or the flag 'End Form On Add' is checked
 - Begin Closing Form
 - Else
 - Begin Clearing Dialog
 7. If the form is in Update Mode
 - If there were no errors attempting to update/add to the database
 - Begin Closing Form

Cancel

Cancel is a standard item that is automatically placed on Fix/Inspect forms.

1. *Perform Event Rules:* Button Clicked
2. *Perform Event Rules:* Post Button Clicked
3. Begin Closing Form

User-defined Items

User-defined items are non-standard items that a developer can add to Fix/Inspect forms to perform specialized processing not handled by the standard items.

1. *Perform Event Rules:* Button Clicked
2. *Perform Event Rules:* Post Button Clicked

Process Flow for Header Detail Form

Description

A Header Detail form is used when a relationship exists between the information in the header and the information contained in the grid. The Header portion uses one business view, and the grid (detail) portion of the form uses another business view. This form type and the Headerless Detail form type are referred to as Transaction forms. The following sections describe the processing flow of a Header Detail form type.

Default Flags

No form option flags are automatically set for this form type. All form option flags can be checked or unchecked by the application developer.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Grid Fields
6. Initialize Static Text
7. Initialize Helps
8. Initialize Event Rules Structures
9. Create Toolbar
10. Load Form Interconnection data into corresponding business view columns and filter fields, if any
11. Perform Event Rules: *Dialog is Initialized*
12. If the form option flag “No Fetch On Form Business View” is checked
 - If the form is not in Add mode and was not entered with a Copy button
 - Perform Event Rules: *Post Dialog is Initialized*
 - Copy the Form Control values to the screen
13. If the form is in Update mode (note that this includes forms entered with a Copy button)

- If the form option flag “No Fetch On Form Business View” is checked
 - Begin Detail Data Selection and Sequencing
 - If the form option flag “No Fetch On Form Business View” is unchecked
 - Begin Header Data Retrieval
14. If the form is in Add Mode
- Begin Clearing Dialog

Header Data Retrieval

A key structure is built for the business view of the header based on the header Business View Columns, or from the filter fields, if any exist. Then a call is made to the database attempting to retrieve the record specified.

1. If the database fetch is successful
 - Copy the fetched database record to the header Business View Columns
 - Perform Event Rules: *Post Dialog is Initialized* (note that where form controls and business view columns have memory, the form controls will have the values from the database during this event which may mean blanks in the case of string and characters)
 - Copy the Business View Columns to the Form Controls
 - If the grid option “Automatically Find on Entry” is checked
 - Begin Detail Data Selection and Sequencing
 - If the grid option “Automatically Find on Entry” is unchecked
 - Begin Add Entry to Row to Grid
2. If the database fetch failed
 - Begin Clearing Dialog.

Detail Data Selection and Sequencing

An internal structure is now created representing the data selection and data sequencing requirements specified by the user for the detail portion. This is then passed to the database engine to do the actual database select and sequencing. The data is retrieved in the next step.

The data used for selection is pulled from filter fields if any exist, or from the key business view columns if there are no filter fields.

1. If the form option flag “No Fetch On Grid Business View” is unchecked
 - Select and Sequence

- Begin Data Retrieval
- 2. If the form option flag 'No Fetch On Grid Business View' is checked
- 3. Begin Add Entry Row To Grid.

Data Retrieval

A request is issued to the JDEKRNL, which performs the actual fetch of the data from the database. It will read one record at a time and for each record, perform the following processing:

1. Attempt to fetch a detail record from the database
2. If a record is fetched
 - Copy the data into the detail business view data structures
 - Perform Event Rules: *Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this grid record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Everest Grid Line - Before*
 - Add the row to the grid. The row now exists in the grid control.
 - Perform Event Rules: *Write Everest Grid Line - After*
 - Clear the grid data structures for reading the next record.
 - Remove the suppress grid line flag.

The previous steps occur for each record read from the database. After all records have been read, the runtime engine will then:

1. If the form is in Copy Mode
 - Switch to Add Mode
 - Begin Clearing Dialog once Detail Data Retrieval processing is complete
2. If no records were fetched
 - Switch to Add Mode
3. Perform Event Rules: *Last Grid Record Has Been Read* (note that this event is run regardless if records were actually fetched)
4. Begin Add Entry Row to Grid.

Clearing Dialog

1. If the form was called in Copy Mode
 - Clear the key controls that have the "Do Not Clear After Add" flag unchecked

2. If the form was not called in Copy Mode
 - Clear all form controls that have the “Do Not Clear After Add” flag unchecked
3. Perform Event Rules: *Clear Screen Before Add*
4. Delete any existing rows
5. Perform Event Rules: *Post Dialog is Initialized*
6. Begin Add Entry Row to Grid

Add Entry Row to Grid

1. Clear grid data structures
2. Perform Event Rules: *Add Last Entry Row to Grid*
3. Add the row to the grid control

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Thread Handling
5. Terminate Helps
6. Free all structures for form, including structures for Business View Columns, Form Controls, Grid Columns, and Event Rules.
7. Destroy the window

Menu/Toolbar Items

OK

OK is a standard item that is automatically placed on header detail forms. It validates the information on the form and updates or adds to the database through JDEKRNL function calls.

1. If there are any errors/warnings on the form, stop OK processing.
2. Perform Event Rules: *Button Clicked*
3. For each control on the form
 - If the current control is a form control and it has not passed validation
 - Perform Event Rules: *Control is Exited*
 - Perform Event Rules: *Control is Exited and Changed - Inline*

- Perform Event Rules: *Control is Exited and Changed - Asynch*
- Perform Data Dictionary validation
- If the current control is a grid control
 - For each grid row

If the current grid row needs to have Leave Row processing run and the row is updateable

Perform Event Rules: *Row is Exited and Changed - Inline* for current row

Perform Event Rules: *Row is Exited and Changed - Asynch* for current row

4. If there are any errors on the form, stop OK processing
5. Delete from the database any grid rows that are in the delete stack. See Delete for details. For each grid row in the delete stack:
 - Copy the information from the delete stack into the grid data structures
 - Copy the grid data structures into the Business View structures
 - Perform Event Rules: *Delete Grid Record from DB - Before*
 - If the database delete has not been suppressed
 - Delete the record in the Business View from the database
 - Delete the grid row from the grid control
 - Perform Event Rules: *Delete Grid Record from DB - After*
6. Perform Event Rules: *All Grid Recs Deleted from DB*
7. If the form option flag 'No Update On Form Business View' is unchecked
 - If the form is in Add mode
 - Perform Event Rules: *Add Record to Database - Before*
 - If the database add has not been suppressed

Add the header business view record to the database
 - Perform Event Rules: *Add Record to Database - After*
 - If the form is in Update mode
 - Perform Event Rules: *Update Record to Database - Before*
 - If the database Update has not been suppressed

Update the record to the database
 - Perform Event Rules: *Update Record to Database - After*

8. If the form option flag 'No Update On Grid Business View' is unchecked
 - For each grid row that was changed or added
 - Clear the business view data structures
 - Reset the original key values for this row in the business view data structures
 - Copy grid data structures to the business view data structures
 - Copy all non-filter database form controls to the business view data structures
 - Copy all equal filters to the business view data structures
 - If form is in Add Mode

Perform Event Rules: *Add Grid Rec to DB - Before*

Add the record in the business view data structure to the database

Perform Event Rules: *Add Grid Rec to DB - After*
 - If form is in Update Mode

Perform Event Rules: *Update Grid Rec to DB - Before*

Update the record in the business view data structure to the database

Perform Event Rules: *Update Grid Rec to DB - After*
 - If form is in Add Mode
 - Perform Event Rules: *All Grid Recs Added to DB*
 - If form is in Update Mode
 - Perform Event Rules: *All Grid Recs Updated to DB*
9. Perform Event Rules: *Post Button Clicked*
10. If form is in Add Mode
 - If form was called in Copy Mode or the flag 'End Form On Add' is checked
 - Begin Closing Form
 - Else
 - Begin Clearing Dialog
11. If the form is in Update Mode
 - If there were no errors attempting to update/add to the database
 - Begin Closing Form

Cancel

Cancel is a standard item that is automatically placed on header detail forms.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*
3. Begin Closing Form

Delete

Delete is a standard item that can be added to Header Detail forms. The delete applies to the grid record(s) selected. The actual delete from the database does not happen at this point, so that if the user clicks cancel, the records are not deleted from the database. The delete verification happens when the delete is pressed, and the actual database delete happens when OK is pressed.

1. Perform Event Rules: *Button Clicked*
2. For each selected grid row that is deleteable
 - Remove Suppress Delete flag
 - Perform Event Rules: *Delete Grid Rec Verify - Before*
 - If the Suppress Delete flag is not set
 - Display Delete Confirmation dialog. If the user clicks NO or CANCEL the rest of this processing is skipped
 - Perform Event Rules: *Delete Grid Rec Verify - After*
 - If the Suppress Delete flag is not set

If the record was read from the database

Add this record to the delete stack (records to be deleted if the user presses OK)

Delete the grid row from the grid control

Find

Find is a standard item that can be added to header detail forms. When clicked by the user, it is the signal to the runtime engine to call the database and reload the grid based on the selections in the filter fields.

1. Perform Event Rules: *Button Clicked*
2. If there no errors in any filter fields
 - Switch to Update Mode
 - Begin Detail Data Selection and Sequencing

- Perform Event Rules: *Post Button Clicked*

Copy

Copy is a standard item that can be added to header detail forms. There is no default processing for copy on header detail forms. They will behave as user-defined items. If a form Interconnection is placed on the Button Clicked event then the called form will open in copy mode.

Add

Add is a standard item that can be added to Header Detail forms. There is no default processing for add on header detail forms. They will behave as user defined items. If a form Interconnection is placed on the *Button Clicked* event, then the called form will open in add mode.

User Defined Items

User-defined items are non-standard items that a developer can add to Header Detail forms to perform specialized processing not handled by the standard items.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*

Process Flow for Headerless Detail Form

Description

A Headerless Detail form is used to update and enter records that have information that is common to all records in a selected group. This form type and the Header Detail form type are referred to as Transaction forms. The following sections describe the processing flow of a Headerless Detail form type.

Default Flags

The following form option flags are automatically checked for Headerless Detail Forms and should not be unchecked: “No Update On Form Business View”, “No Fetch On Form Business View”. Other form option flags can be checked or unchecked by the application developer.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Grid Fields
6. Initialize Static Text
7. Initialize Helps
8. Initialize Event Rules Structures
9. Create Toolbar
10. Load Form Interconnection data into corresponding business view columns and filter fields, if any
11. Perform Event Rules: *Dialog is Initialized*
12. If the form is not in Add Mode
 - If the form is not in Copy Mode
 - Perform Event Rules: *Post Dialog is Initialized* (note that this event is run immediately after Dialog is Initialized when in Update mode, so the FC values are still at their NULL or zero value)
 - If the grid option “Automatically Find on Entry” is checked

- Begin Data Selection and Sequencing
- If the grid option “Automatically Find on Entry” is unchecked
 - Begin Add Entry Row to Grid
- If the form is in Add Mode
 - Begin Clearing Dialog

Data Selection and Sequencing

An internal structure is now created representing the data selection and data sequencing requirements specified by the user. This is then passed to the database engine to do the actual database select and sequencing. The data is then held until the data is retrieved in the next step.

The data used for selection is pulled from filter fields if any exist, or from the key business view columns if there are no filter fields.

1. If the form option flag “No Fetch On Grid Business View” is unchecked
 - Select and Sequence
 - Begin Data Retrieval
2. If the form option flag “No Fetch On Grid Business View” is checked
 - Begin Add Entry Row To Grid

Data Retrieval

A request is issued to the JDEKRNL, which performs the actual fetch of the data from the database. It will read one record at a time and for each record, perform the following processing:

1. Attempt to fetch a record from the database
2. If a record is fetched
 - Copy the data into the business view data structures
 - Perform Event Rules: *Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this grid record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Grid Line - Before*
 - Add the row to the grid. The row now exists in the grid control.
 - Perform Event Rules: *Write Grid Line - After*
 - Clear the grid data structures for reading the next record.
 - Remove the suppress grid line flag.

The previous steps occur for each record read from the database. After all records have been read, the runtime engine will then:

1. If the form is in Copy Mode
 - Switch to Add Mode
 - Begin Clearing Dialog once Data Retrieval processing is complete
2. If no records were fetched
 - Switch to Add Mode
3. Perform Event Rules: *Last Grid Record Has Been Read* (note that this event is run regardless of whether records were actually fetched)
4. Begin Add Entry Row to Grid.

Clearing Dialog

1. If the form was called in Copy Mode
 - Clear the key controls that have the “Do Not Clear After Add” flag unchecked
2. If the form was not called in Copy Mode
 - Clear all form controls that have the “Do Not Clear After Add” flag unchecked
3. Perform Event Rules: *Clear Screen Before Add*
 - Delete any existing grid rows
4. Perform Event Rules: *Post Dialog is Initialized*
5. Begin Add Entry Row to Grid

Add Entry Row to Grid

1. Clear grid data structures
2. Perform Event Rules: *Add Last Entry Row to Grid*
3. Add the row to the grid control

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Thread Handling
5. Terminate Helps

6. Free all structures for form, including structures for Business View Columns, Form Controls, Grid Columns, and Event Rules
7. Destroy the window

Menu/Toolbar Items

OK

OK is a standard item that is automatically placed on Headerless Detail forms. It validates the information on the form and updates or adds to the database through JDEKRNL function calls.

1. If there are any errors/warnings on the form, stop OK processing
2. *Perform Event Rules: Button Clicked*
3. For each control on the form
 - If the current control is a form control and it has not passed validation
 - Perform Event Rules: *Control is Exited*
 - Perform Event Rules: *Control is Exited and Changed - Inline*
 - Perform Event Rules: *Control is Exited and Changed - Asynch*
 - Perform Data Dictionary validation
 - If the current control is a grid control
 - For each grid row
 - If the current grid row needs to have Leave Row processing run and is updateable
 - Perform Event Rules: *Row is Exited and Changed - Inline* for current row
 - Perform Event Rules: *Row is Exited and Changed - Asynch* for current row
4. If there are any errors on the form, stop OK processing
5. Delete from the database any grid rows that are in the delete stack. See [Delete](#) for details. For each grid row in the delete stack:
 - Copy the grid row data into the Business View structures
 - Copy the grid data structures into the business view data structures
 - Perform Event Rules: *Delete Grid Record from DB - Before*
 - If the database delete has not been suppressed
 - Delete the record in the Business View from the database
 - Delete the grid row from the grid control

- Perform Event Rules: *Delete Grid Record from DB - After*
- 6. Perform Event Rules: *All Grid Recs Deleted from DB*
- 7. If the form option flag 'No Update On Grid Business View' is unchecked
 - For each grid row that was changed or added
 - Clear the business view data structures
 - Reset the original key values for this row in the business view data structures
 - Copy grid data structures to the business view data structures
 - Copy all non-filter database form controls to the business view data structures
 - Copy all equal filters to the business view data structures
 - If form is in Add Mode

Perform Event Rules: *Add Grid Rec to DB - Before*

Add the record in the business view data structure to the database

Perform Event Rules: *Add Grid Rec to DB - After*
 - If form is in Update Mode

Perform Event Rules: *Update Grid Rec to DB - Before*

Update the record in the business view data structure to the database

Perform Event Rules: *Update Grid Rec to DB - After*
 - If form is in Add Mode
 - Perform Event Rules: *All Grid Recs Added to DB*
 - If form is in Update Mode
 - Perform Event Rules: *All Grid Recs Updated to DB*
- 8. Perform Event Rules: *Post Button Clicked*
- 9. If form is in Add Mode
 - If form was called in Copy Mode or the flag 'End Form On Add' is checked
 - Begin Closing Form
 - Else
 - Begin Clearing Dialog
- 10. If the form is in Update Mode
 - If there were no errors attempting to update/add to the database
 - Begin Closing Form

Cancel

Cancel is a standard item that is automatically placed on Headerless Detail forms.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*
3. Begin Closing Form

Delete

Delete is a standard item that can be added to Headerless Detail forms. The actual delete from the database does not happen at this point, so that if the user clicks cancel, the records are not deleted from the database. The delete verification happens when delete is pressed, and the actual database delete happens when OK is pressed.

1. Perform Event Rules: *Button Clicked*
2. For each selected grid row that is deleteable
 - Remove Suppress Delete flag
 - Perform Event Rules: *Delete Grid Rec Verify - Before*
 - If the Suppress Delete flag is not set
 - Display Delete Confirmation dialog. If the user clicks NO or CANCEL the rest of this processing is skipped.
 - Perform Event Rules: *Delete Grid Rec Verify - After*
 - If the Suppress Delete flag is not set
 - If the record was read from the database
 - Add this record to the delete stack (records to be deleted if the user presses OK)
 - Delete the grid row from the grid control

Find

Find is a standard item that can be added to Headerless Detail forms. When clicked by the user, it is the signal to the runtime engine to call the database and reload the grid based on the selections in the Filter Fields.

1. Perform Event Rules: *Button Clicked*
2. If there no errors in any filter fields
 - Switch to Update Mode
 - Begin Data Selection and Sequencing
 - Perform Event Rules: *Post Button Clicked*

Copy

Copy is a standard item that can be added to Headerless Detail forms. There is no default processing for copy on Headerless Detail forms. They will behave as user-defined items. If a form interconnection is placed on the *Button Clicked* event then the called form will open in copy mode.

Add

Add is a standard item that can be added to Headerless Detail forms. There is no default processing for add on Headerless Detail forms. They will behave as user defined items. If a form interconnection is placed on the *Button Clicked* event then the called form will open in add mode.

User Defined Items

User-defined items are non-standard items that a developer can add to Headerless Detail forms to perform specialized processing not handled by the standard items.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*

Process Flow for Search/Select Form

Description

A Search/Select form is used to select a single predetermined field from a record in a predetermined file. It is extremely important to note that Search/Select forms return only one value to the calling form, based on the dictionary alias. If there is no data dictionary alias match, the first value from the data structure will be returned. The following sections describe the processing flow of a Search/Select form type. A Search/Select form should only be attached as a visual assist to data items that have the same data type as the form data structure element.

Default Flags

No form option flags are automatically set for this form type, but the Grid option flag 'Automatically Find on Entry' is often checked by application developers. The only form option flag that impacts this form type is the "No Fetch on Grid Business View" flag. Checking any of the other form option flags does not have any impact on the form processing.

Dialog Initialization

1. Initialize Thread Handling
2. Initialize Error Handling Process
3. Initialize Business View Columns
4. Initialize Form Controls
5. Initialize Grid Fields
6. Initialize Static Text
7. Initialize Helps
8. Initialize Event Rules Structures
9. Create Toolbar
10. Load Form Interconnection data into corresponding business view columns and filter fields, if any
11. Perform Event Rules: *Dialog is Initialized*
12. Perform Event Rules: *Post Dialog is Initialized*
13. If the grid option "Automatically Find On Entry" is checked
 - Begin Detail Data Selection and Sequencing

Header Data Retrieval

There is no header record on Search/Select Forms.

Detail Data Selection and Sequencing

An internal structure is now created representing the data selection and data sequencing requirements specified by the user. This is then passed to the database engine to do the actual database select and sequencing. The data is then held until the data is retrieved in the next step.

The data used for selection is pulled from filter fields.

1. If the form option flag “No Fetch On Grid Business View” is unchecked
 - Select and Sequence
 - Begin Data Retrieval

Data Retrieval

A request is issued to the JDEKRN, which performs the actual fetch of the data from the database. It will read one record at a time and for each record, perform the following processing:

1. Attempt to fetch a record from the database
2. If a record is fetched
 - Copy the data into the business view data structures
 - Perform Event Rules: *Grid Record is Fetched*
 - If the application developer has not chosen to suppress the writing of this grid record
 - Copy the business view data into the grid data structures
 - Perform Event Rules: *Write Grid Line - Before*
 - Add the row to the grid. The row now exists in the grid control
 - Perform Event Rules: *Write Grid Line - After*
 - Clear the grid data structures for reading the next record
 - Remove the suppress grid line flag

The previous steps occur for each record read from the database. The following occurs only once.

- Perform Event Rules: *Last Grid Record Has Been Read*

Clearing Dialog

Not done for this form type.

Closing Form

1. Perform Event Rules: *End Dialog*
2. Load Form Interconnection data from corresponding business view columns, if any
3. Terminate Error Handling
4. Terminate Helps
5. Free all structures for form, including structures for Business View Columns, Form Controls, Grid Columns, and Event Rules
6. Destroy the window

Menu/Toolbar Items

Select

Select is a standard item that is automatically placed on Search/Select forms. It returns a value to the form interconnection and closes the form.

1. Copy the selected grid row into the business view column
2. Perform Event Rules: *Button Clicked*
3. Perform Event Rules: *Post Button Clicked*
4. Begin Closing Form

Close

Close is a standard item that is automatically placed on Search/Select forms.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*
3. Begin Closing Form

Find

Find is a standard item that can be added to Search/Select forms. When clicked by the user, it is the signal to the runtime engine to call the database and reload the grid based on the selections in the Filter Fields.

1. Perform Event Rules: *Button Clicked*

2. Begin Data Selection and Sequencing
3. Perform Event Rules: *Post Button Clicked*

Delete

Delete is a standard item that can be added to Search/Select forms. There is no default processing for delete on Search/Select forms. They will behave as user-defined buttons.

Copy

Copy is a standard item that can be added to Search/Select forms. There is no default processing for copy on Search/Select forms. They will behave as user-defined buttons.

Add

Add is a standard item that can be added to Search/Select forms. There is no default processing for add on Search/Select forms. They will behave as user defined buttons.

User-defined Items

User-defined items are non-standard items that a developer can add to Search/Select forms to perform specialized processing not handled by the standard buttons.

1. Perform Event Rules: *Button Clicked*
2. Perform Event Rules: *Post Button Clicked*

Process Flow for Message Form

Description

The Message form type is a form that comes up as a secondary window to inform the user of something or to ask a question. It parallels the behavior of a Windows message box. The form does not have a toolbar or a status bar and can only contain static text and buttons. It is the only form type that can have standard buttons appearing on the form instead of the toolbar.

Dialog Initialization

1. Initialize Form Controls
2. Initialize Static Text
3. Initialize Helps
4. Initialize Event Rules Structures
5. Perform Event Rules: *Dialog is Initialized*

Closing Form

1. Terminate Helps
2. Free all structures for Controls and Event Rules.
3. Destroy the window

Buttons

The form has an OK button by default. Other options include Cancel, Yes, and No. All of these buttons will cause the dialog to close, and there is no other default processing.

J.D. Edwards Standards

These forms may only have static text and buttons. Event rules are limited to only *Dialog is Initialized* and any button events. This event rule may not contain any Form Interconnection calls.

Process Flow for Edit Control

Description

An edit control is a text field on a form. All form types can contain edit controls except Message forms. There are two types of edit controls. The first type is commonly referred to as a database field. It is associated with an item in the business view and through that connection to a dictionary item. Database fields represent a field in a database record. The second type of edit control is commonly referred to as a data dictionary field. Although only one type is called a data dictionary field, both types have a connection to a specific data dictionary item.

Within the realm of database fields there is an added distinction between filter fields and non-filter fields. Database fields are nonfilter fields by default. A filter field is used to alter the selection criteria of a database fetch. A filter can have a comparison type of equal, not equal, less than, less than or equal to, greater than, or greater than or equal to. A filter can be marked so that a wildcard (*) displays when the filter is not included in the selection.

The actual storage for the edit control's value is based on the type of the associated dictionary item (for example math numeric, string, character) and is distinct from the screen representation of the value. An edit control is affected by the properties of the associated dictionary item. For example, if an edit control is associated with a dictionary item of type string with a length of thirty, that edit control will not allow more than thirty characters to be typed into the field.

Properties and Options

The following properties and options are available for edit controls. These properties are available on all form types and for both database fields and data dictionary fields. Only those properties that affect processing are discussed here. Those properties that are exclusively interface-related are either not discussed, or are only discussed as they relate to control processing.

Disable

Controls with this property appear gray and cannot be changed by the user. You can use event rules to change the value of disabled fields. When a control is disabled, the associated text (static text to the left) is also disabled. Controls do not have this property by default.

Visible	Controls with this property can be seen. Without this property the control cannot be seen. You can use event rules to change the value of hidden controls.
Do not clear after add	This option applies only when a form is in add mode and the form is being cleared. Controls with this option retain their value. Controls do not have this option by default.
Required entry field	This option requires that a value is entered in the control before the record is saved (OK processing). This option does not accept either Null or blank. Controls do not have this option by default.
Default cursor on add/update mode	This option specifies where the cursor will be placed in add/update mode. If multiple controls have this property on, you cannot predict which control will actually have the cursor. Controls do not have this option by default.
No display if currency is Off	This option causes the control to be hidden when the Multicurrency Conversion is off. Controls do not have this option by default.

Control is Entered

1. Perform event rules: *Control is Entered*

On a Windows client, when a user presses the tab key or otherwise causes focus to move from one control to another, the control receiving focus actually gets the “Control is Entered” message before the control losing focus gets the “Control is Exited” message. Because this is a direct result of the sequence of Windows messages, it cannot be altered, although it is somewhat counterintuitive.

Control is Exited

1. If the focus is going to another window, stop processing (focus will return to this control when this window again receives focus)
2. Copy the text from the screen to the internal storage (conversion from string to the appropriate type happens during this step)
3. If this control is marked as a Required Entry Field and does not contain a value, then set the field in error.
4. Perform event rules: *Control is Exited*
5. If this is the first time this control has been exited, or if this control has changed in value since the last time the control was exited, then:
 - Begin Control is Exited and Changed (*Inline*)

Control is Exited and Changed (Inline portion)

1. Perform event rules: *Control is Exited and Changed Inline*
2. If the form is not closing:
 - Attempt to execute *Control is Exited and Changed* (Asynchronous) on a thread.
3. If the form is closing or if the attempt to execute asynchronously was unsuccessful:
 - Begin *Control is Excited and Changed* (Asynchronous) as an inline function.

Control is Exited and Changed (Asynchronous portion)

1. Perform event rules: *Control is Exited and Changed Asynch*
2. If no errors were set during event rules processing
 - If this is not a filter field, or if it is an equal filter field (validation takes place only on filter fields with a comparison type of equal):
 - Perform Data Dictionary Validation
 - If there is an associated description for this control, populate it (if there were errors, this clears the associated description)
 - If there were no errors during validation, then copy the value to the control

J.D. Edwards Standards

All filter fields are marked with “Display Wildcard.” Only index fields are marked as filter fields (for performance). If reasonable, event rule should be put on *Control is Exited and Changed Asynch* instead of on one of the inline events (for performance).

Process Flow for Grid Control

Description

A grid control is similar to a spreadsheet on a form. Find/Browse, Search and Select, Header Detail, and Headerless Detail form types can contain grid controls. Grid controls contain columns. The columns are specified at design time. There are two types of columns. The first type is commonly referred to as a database column. It is associated with an item in the business view and through that connection to a dictionary item. Database columns represent a field in a database record. The second type of column is commonly referred to as a data dictionary column. Although only one type is called a data dictionary column, both types have a connection to a specific data dictionary item. The difference is that a database column has the additional connection to a business view field.

A grid can either be a browse grid or an update grid. You can use a browse grid is for viewing only and cells cannot be selected individually. The Find/Browse form and the Search and Select form have browse grids. You can use an update grid to add or update records. Cells in an update grid can be selected individually. The Header Detail form and the Headerless Detail form have update grids.

Grid controls can also have a Query By Example (QBE) line. The QBE columns have a one to one correspondence with the grid columns. You use a QBE value to change the selection criteria of a database fetch. Only database columns allow entry in the QBE columns because the purpose of the QBE is to affect the select and only database columns are in the business view. A QBE column can have one of the following comparison types:

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

The comparison type is equal unless you specify otherwise. You can specify the comparison type in the QBE column or by using system functions. You can use wildcards (* or %) for an inexact search on a string field.

The values contained on a grid row act as a logical unit. You must validate a grid row prior to accepting a record, but validating the individual columns is not required. There is a parallel between edit control validation and grid row validation. There is not a

parallel between edit control validation and grid column validation. The *Column is Exited* events are executed only if the user physically exits the cell. The Data Dictionary Validation for a cell executes when the cell is exited after a change or the first time the row is exited after it has been added. If you change a cell programmatically, then the *Row is Exited* events execute prior to accepting a record, but the column events and validation do not execute unless focus is physically set on the column.

The vendor spreadsheet stores the grid cell values as a tab delimited string (one per row). The values can be retrieved on a cell by cell basis or on a row by row basis. There is also internal storage for the grid columns in the interactive engine. The actual storage for the grid columns value is based on the type of the associated dictionary item (for example, math numeric, string, character) and is distinct from the screen representation of the value. Only one row of data can be acted upon at any given time. Each event executes in the context of a specific row.

A grid column is affected by the properties of the associated dictionary item. For example, if a grid column is associated with a dictionary item of type string with a length of 30 that grid column will not allow more than 30 characters to be typed into the cell.

Grid Columns, and Query By Example (QBE) lines are discussed in separate flow documents, but are an integrated portion of the grid. [Grid Column link](#), [QBE link](#).

Properties and Options

The following properties and options are available for grid controls. Except where noted these properties are available on all grids. Only those properties that affect processing are listed here. Those properties that are exclusively interface related are either not discussed, or are only discussed as they relate to grid processing. For a more detailed discussion of the interface properties reference the Forms Design Aid documentation.

Disable	Grids with this property appear grey and cannot be changed by the user. The application developer can change the value of disabled fields through ER. Grids do not have this property by default.
Visible	Grids with this property can be seen. Without this property the grid cannot be seen. The application developer can cause hidden grids to change value through ER.
Hide Query By Example	Grids with this property do not have a QBE line. QBE is neither available to the user nor the ER. Browse grids do not have this property by default. Update grids have this property by default.

Update Mode	This property is only available on update grids, but does not have any effect on the grid during runtime.
Multiple Select	this property allows for multiple grid rows to be selected at once. this can impact ER execution and deleting. Grids do not have this option by default.
Automatically Find On Entry	This option determines if grid records will be fetched when the form is opened. Grids without this option will open with no grid rows. Grids do not have this property by default.
Auto Find On Changes	This option determines if grid records will be fetched after a child form that has changed records closes. this option should only be used on forms where there are no modeless form interconnects. Grids do not have this property by default.
No Adds on Update Grid	This property applies to updates only. It determines if there will be an entry row in the grid. Without an entry row records cannot be added. With this property checked the only records that can be altered are existing records (records can be updated only). Grids do not have this option by default.
Disable Page-At-A-Time Processing	This option causes all available grid records to be fetched with find is pressed. Without this option only the first page of grid records is fetched until the user scrolls down to see additional records. Each time more records are requested, a page of data is returned. This provides a substantial performance benefit for large files and it is not recommended that this option be checked without careful consideration. Grids do not have this option by default.
Clear Grid After Add	There is currently no reference to this field during runtime. Presence or absence of this flag has no effect on the grid
Refresh Grid After Update	There is currently no reference to this field during runtime. Presence or absence of this flag has no effect on the grid
Process All Rows In Grid	This option causes each row in the grid to perform the three Row is Exited events on all the grid rows at least once prior to updating or adding the database records.

Set Focus on Grid

1. Perform event rules: *Set Focus on Grid*

Kill Focus on Grid

1. Perform event rules: *Kill Focus on Grid*

Row is Entered

1. Update the status bar with the current row number
2. If this row is not the entry row (last row on update grids):
 - Perform event rules: *Row is entered*

Row is Exited

1. If the form is not loosing focus:
 - Perform event rules: *Row is exited*
 - If this is an update grid and the row has been changed since the last time the row was exited
 - Perform event rules: *Row is Exited and changed - Inline*
 - If you are not leaving the grid (exiting one row, entering another):
 - Attempt to execute Row is Exited and Changed (Asynchronous portion) on a thread.
 - If we are leaving the grid or the attempt to execute asynchronously was unsuccessful:
 - Begin Row is Exited and Changed (Asynchronous portion) as an inline function.

Row is Exited and Changed (Asynchronous portion)

1. Set a bitmap on the grid row header to indicate that this row is being processed.
2. Perform event rules: *Row is Exited and Changed - Asynch*
3. If this is the first time this row has been exited since it was added:
 - Begin Row is Exited Validation

Row is Exited Validation

For each grid cell in this row that has not already performed data dictionary validation:

1. If this data base item is also in the header portion of the form the grid column will be populated from there, so skip this validation and go to the next grid cell. Note: Filter fields are only populated to the grid column if they are equal filters.
2. If the text contained in the cell can be stored (no alphas in numerics, no out or range data parameters):

- Begin Data Dictionary Validation for this cell.
- If there is an associated description column for this cell, populate it with the information returned from Data Dictionary Validation.

Cell is Exited after a change

On update grids when the contents of a cell have changed (via user keying or visual assist) and the cell has been exited:

1. Clear any errors set on this cell previous calls to Row is Exited Validation and Column is Exited events.
2. If the text contained in the cell can be stored (no alphas in numerics, no out of range date parameters):
 - Perform Data Dictionary Validation
 - If there is an associated description column for this cell, populate it with the information returned from Data Dictionary Validation.

Double Click on a Grid Row

On browse grids, double clicking on the grid row will cause the Select button to be pressed.

Key Pressed

On update grids when a key is pressed on the entry row (last row in the grid), it causes a new row to be added to the grid.

J.D. Edwards Standards

The event Row is Exited and Changed - Asynch is equivalent to validating the rows contents. it is often used for the "Edit Line" master business function. If reasonable, ER should be put on Row is Exited and Changed - Asynch instead of on one of the inline events (performance).



Appendix C: Smart Fields

Smart Fields are actually data dictionary items with attached business functions. The business functions that are attached include named mappings. This makes selecting a data item with particular functionality much easier. Instead of the end-user having to know which business function to use and what parameters to pass, the user simply has to select a data item that inherently has this information. Smart fields can be used for things like deriving a column heading or an object value in a tabular section in Report Design.

This section describes the following topics:

- ☐ Understanding Smart Fields
- ☐ Creating Smart Fields



Understanding Smart Fields

Smart fields are reusable objects that simplify using business functions in event rules. They are data dictionary items with business functions attached. You create a smart field to perform a specific function. For example, you can create a smart field to add period 1, period 2, and period 3 together to equal quarter 1. You want to perform this calculation for the column titled Quarter 1 each time a row of data is read. You put the smart field on the report so that the user does not need to put this calculation on each column. Report Design uses the smart field to do the calculation automatically.

Smart fields are only available for columns in tabular section reports. In addition to defining the smart field, you must also have a column heading and data selection.

The basic steps to create a smart field include:

- Creating a business function
- Creating a named mapping
- Creating a smart field data item

Refer to *Creating Smart Fields* for more information about creating a smart field.

Once the appropriate smart fields exist, you can create a smart field template in Report Design. The smart field template and the Configurable Director allow you to organize and present your smart fields to be easily used by the user. You can use the Configurable Director to help you define things like report processing options, properties, or business views for a report. You can use a smart field template to collect a group of compatible smart fields.

See Also

- *The Enterprise Reporting Guide*

Named Mappings

You use named mappings to map the source for each parameter of the business function data structure. The values used for smart fields may be known to a business function without requiring user input. For example, the desired fiscal year or period may already be specified in processing options. The named mapping for the business function holds the default values to be used for the business function connection in event rules so that the user does not need to be prompted for these values.

Smart Field Prompts

You cannot always predefine all of the information required to set up smart fields. Some information requires user input. You use prompts to give the user the flexibility to define offsets for things like the current period and fiscal year. For example, if you choose the Account Balance smart field, you are prompted for period offsets, fiscal year offsets, and column headings.

Defining Data Selection

When you define data selection for a tabular report, you can choose multiple companies, ledger types, or periods of fiscal years. The data selection for the report must include all of the data for all of the columns and rows. For example, in the following report example the data selection includes actual amounts for both this year and last year for all companies. When the column for this year is calculated, the data for that column is refined to “this year” based on the data selection over the column. When the first row is processed, only the data for Company 1 is included.

	Actual Amount This Year	Actual Amount Last Year
Company 1	9999	9999
Company 2	9999	9999

Smart Field Template

You can use a smart field template to group the smart fields you want include in a report director. For example, you can create a financial reporting smart field template that includes all of the smart fields for financial reporting and use this template for a financial reporting director. For more information about smart field templates and report director templates refer to *Working with Report Director Templates* in the *Enterprise Reporting Guide*.

Example: Smart Field

You can use smart fields to determine column headings and values. For example, suppose you want a report that contains the following data:

- company
- ledger type
- fiscal year
- amount

	Quarter 1	1998 Budget Amount	1997 Actual Amount	1997 Budget Amount
Company 1	9999	9999	9999	9999
Company 2	9999	9999	9999	9999

For this example, you want to use this same report for years into the future without having to maintain it. This means that the column heading must change to match the data you are using. You can use smart fields that are included with Report Design or you can create new ones to meet your needs. You can use a smart field to display the current fiscal year or the previous fiscal year. You can use the business function attached to the smart field to determine the current fiscal year and make sure that the data coming into the report matches the year. If the years do not match, the value is not accumulated. If the year does match, then the values are accumulated. The smart field can also update the column headings based on the the data being displayed.

Creating Smart Fields

There are several tasks you must complete to create a smart field.

- Create a business function (this requires a data structure)

For example, you might want a business function to manipulate the amount and the column headings. You could also use a business function to decide whether the row fetched has the fiscal year you want based on the company in the fetched row.

For information about creating a business function refer to *Business Functions* in this guide.

- Create a named mapping using the data structure from the business function.

For more information about data structures refer to *Data Structures* in this guide.

- Create a smart field data item

The smart field data item is associated with a named mapping and a business function. Refer to *Creating a Smart Field Trigger* for more information about creating a smart field data item.

Before You Begin

Before you create a smart field you should be familiar with the following:

- ☐ Report Design, including tabular sections and using smart fields
- ☐ Business Views
- ☐ Business Functions

Creating a Named Mapping

You can create named mappings for business function data structures. Named mappings are used for smart fields. They hold the default source values for smart field parameters. If you create a named mapping for a business function data structure, the named mapping associations are checked in and out when you check the data structure in or out.

► To create a named mapping

On Object Librarian

1. Enter DSTR in the Object Type field.
2. Choose the business function data structure for which you wish to create a named mapping.

Name	Description	Ins Sys	Rep Sys	Object Type	Object Use	Related Object	Function Type
B0002	BSFN Data Structure	42	42	DSTR			
B4201070D	Cache, Update Cust. Self-Service Information	42	42	DSTR			
D0000142	Get Cost Center Description A7	42	42	DSTR			
D4002060D	Get Preference Invoice Cycle	42	42	DSTR	330		
D4002290	F4209 Verify Order For Holds	42	42	DSTR	330		
D4002410	Retrieve or change string element	42	42	DSTR			
D4100380D	Update Multiple Return	42	42	DSTR			
D4200010	Load F4201 Structure	42	42	DSTR			
D4200020	Load F4211 Structure	42	42	DSTR			
D4200030	F4201 Get Order No. From Next Number	42	42	DSTR			
D4200040	F420003 Verify Commission Code	42	42	DSTR			
D4200050	F42004 Verify Commission Code	42	42	DSTR			

3. On Parameter Design, choose Named Map from the Form menu.

Location	Path Code	S C	SAR	User ID	Date Modified	Time Modified	Modification Comment
INTELNTC	APPL_PGF	1	1882463	AH5702757	1/8/98	10:54:02	
INTELNTC	PROD	1		JC5482504	2/25/98	10:20:24	
SAYLERB1	APPL_PGF	3	1882463	BS5625464	8/2/98	16:58:49	

- On Named Mapping click Add.

Named Mapping

Named Mapping List

☐ Default

Name	Data Item	Req	I/O
mnCommissionCode	DocumentOrderInvoiceE	☐	☒
cErrorCode	ErrorCode	☐	☒
cCallType	EverestEventPoint01	☒	☒

Data Item Origin

Origin Types

☒ Literal
☐ Prompt
☐ Table
☐ Data Dictionary Item
☐ System Value

Value

Table

☐ Required ☐ Return Value

- Complete the following field:

The named mapping appears in the Named Mapping List field. Named Mapping List displays all existing named mappings.

A business function data structure can have more than one named mapping, but a smart field only uses one named mapping.

- Turn on the Default option if you wish to make this the default mapping. If you do not select a specific mapping for the default, the one at the top of the list box will automatically become the default.

Named Mapping

Named Mapping List

☒ Default

Name	Data Item	Req	I/O
mnCommissionCode	DocumentOrderInvoiceE	☐	☒
cErrorCode	ErrorCode	☐	☒
cCallType	EverestEventPoint01	☒	☒

Data Item Origin

Origin Types

☒ Literal
☐ Prompt
☐ Table
☐ Data Dictionary Item
☐ System Value

Value

Table

☐ Required ☐ Return Value

- Click one of the following Data Item Origin options.

You can browse for the name of the item you wish to use.

Literal	Enter the literal value to assign to the data item. This option displays an alphanumeric field. Users will enter the correct value for the data item being mapped.
Prompt	Enter the data item to be used when prompting the report designer for a value. The text for the prompt will be the description and glossary text for the specified item.
Table	Enter a table name and the data item within the table selected.
Processing	Enter the processing option data item name. This is used globally within the scope of the application. If the same data item is used more than once in the processing option template, the first matching data item will be used.
System Value	Enter the system value data item name that is used throughout the entire system. System values are fetched from the F98VAR table using the filter OneWorld data type is equal to the OneWorld data type of the data item, and the variable type is equal to the system_value. An example is system date.

The Named Mapping form changes depending on the data required for the Data Item Origin you choose.

Named Mapping

Named Mapping List: ☒ Default

Name	Data Item	Req	I/O
mnComissionCode	DocumentOrderInvoiceE	☒	☒
cErrorCode	ErrorCode	☒	☒
cCallType	EverestEventPoint01	☒	☒

Data Item Origin

Origin Types:

- ☐ Literal
- ☐ Prompt
- ☐ Table
- ☒ Data Dictionary Item
- ☐ System Value

Optional:

- ☒ All
- ☐ Report Variable
- ☐ Report Constant
- ☐ Processing Option & Report Interconnect

Value:

Table:

☐ Required ☐ Return Value

8. You must choose each data item and enter data into both Origin type and Name until each data item column is filled.
9. Click OK.

► To change a named mapping

On Named Mapping

1. Choose the data item you wish to change and make any appropriate changes.
2. Make sure at least one Data Item is set as a return value.

If a Data Item is set to be returned, it is also required.

3. Make sure at least one Data Item is not set as “Prompt”, because “Prompt” cannot have a return value.

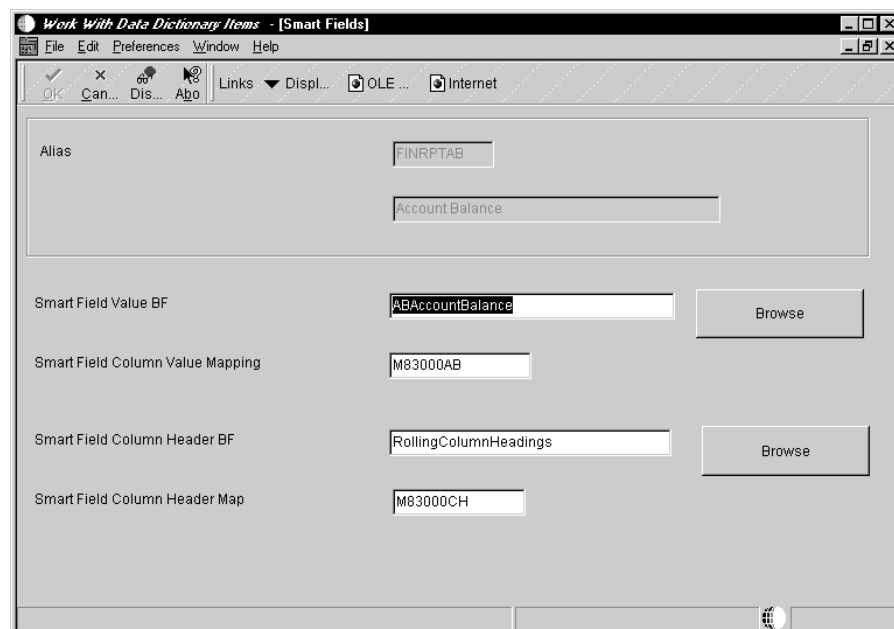
► To create a smart field data item

1. Create a data dictionary item.

Use glossary group K for smart fields.

See *Defining a Data Item* for information about creating data dictionary items.

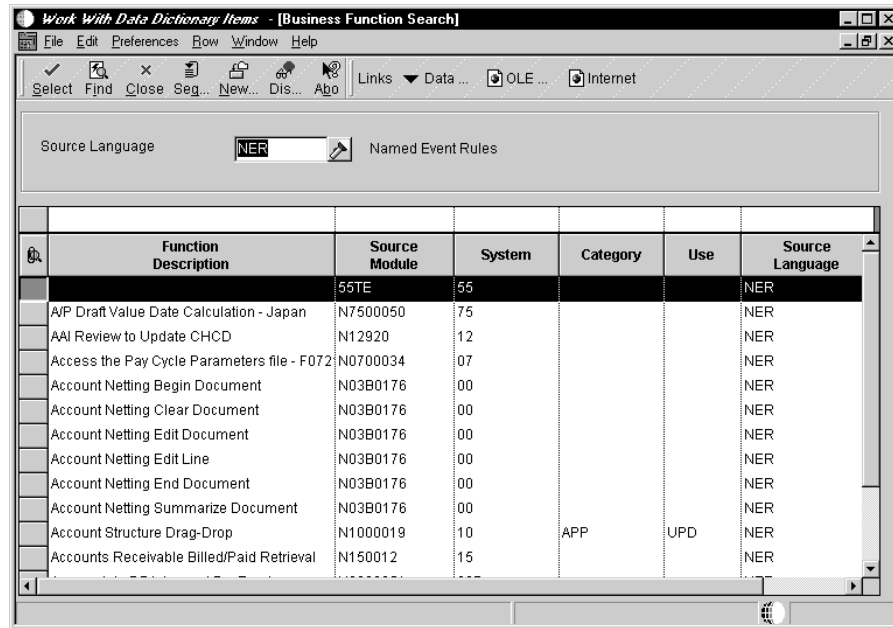
2. On Data Item Specifications, click the Smart Field button.



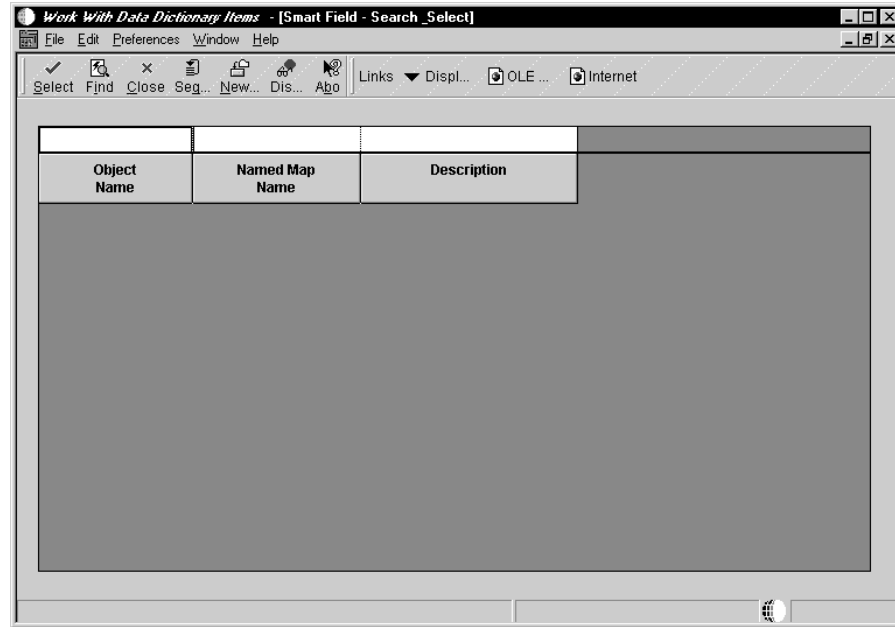
3. On Smart Fields, complete one or more of the following fields:

- Smart Field Value BF
- Smart Field Column Value Mapping
- Smart Field Column Header BF
- Smart Field Column Header Map

On Smart Fields, you can click the Browse button to select a business function.



On Smart Fields, you can click the Visual Assist on Smart Field Column Value Mapping to select a value.



Once a smart field is created, it may be used for creating reports.

See Also

- For information about using smart fields see the *Enterprise Report Writing Guide*.



Appendix D: Date Reference Scan

You can run date scan programs to search for date references in event rules or business functions. This may be helpful if you wish to verify possible Year 2000 issues.



Working with the Date Reference Scan

J.D. Edwards provides two date reference scans. You can use the

- ☐ Business Function Date Reference Scan
- ☐ Event Rule Date Reference Scan

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 check the reports that are created during this process. J.D. Edwards provides the following OneWorld date reference scans.

- Business function date references scan
- Event rule date references scan

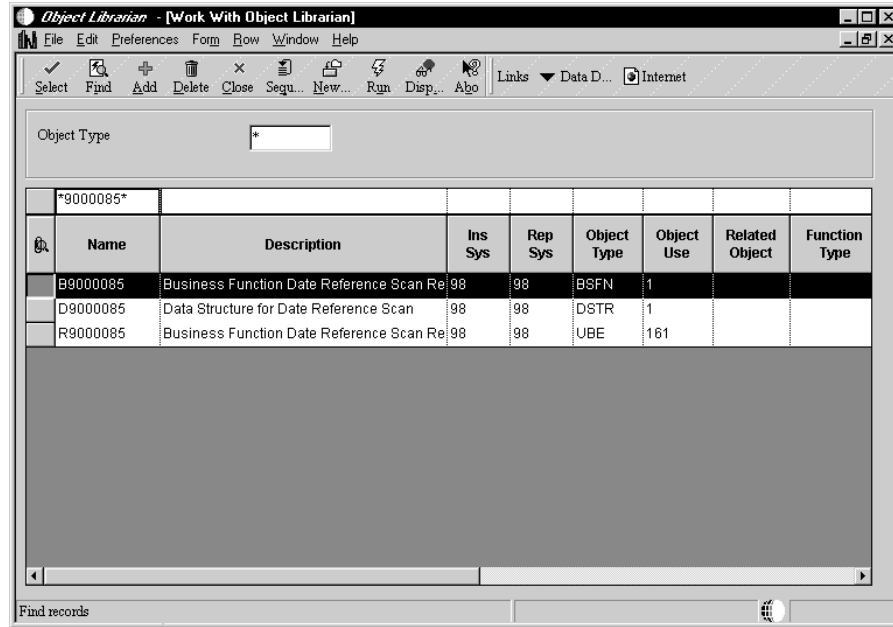


To use the business function date reference scan

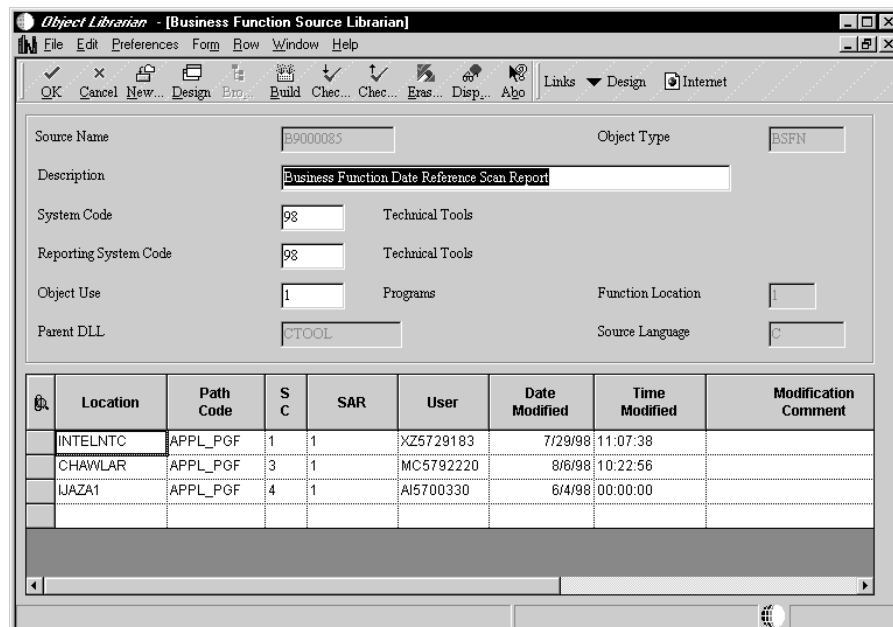
On Object Librarian

1. Check out the following objects.
 - B9000085
 - D9000085
 - R9000085

This step brings the specifications you need to your workstation.



- Choose B9000085 and click Select.



- On Business Function Source Librarian, click the Build button.
- From menu GH9011, choose Batch Versions (P98305).
- On Work with Batch Versions, choose R9000085.

Location	Path Code	User	Checked Out	On Server	Server Last Updated
INTELNTC	APPL_PGF	XZ5729183	N	Y	5/28/98

6. Choose the version you wish to run.
7. On Data Selection and Sequencing choose which option you wish to use and click Submit.

☒ Data Selection ☐ Data Sequencing

8. On Criteria Design designate the criteria you wish to use for your report and click OK.
9. Check your PDF report and the text log file in the default print directory, which is usually the B7\PrintQueue directory.

When you run R9000085 it creates two reports. One report has the number of date references the scan program detected. The other report shows the date reference details.

Event Rule Date Reference Scan

You can use an event rule date scan to check event rules for date references. You can then check the reports that are created during this process.

The process to do this is the same as the business function scan except you use the following objects.

- B9000080
- D9000080
- R9000080

When you run R9000080 it creates two reports. One report has the number of date references the scan program detected.

The following illustration is an example of an event rule date reference report.

Event Code	Obj Name	Event Rules date references	scan program Member Description	No of date references	Date Scan Result
00	APPL	P0000	System Setup	5	Date references found
00	APPL	P0001	Company/Business Unit Tree Structure	0	Date references found
00	APPL	P0002	Next Numbers	32	Date references found
H05	APPL	P00022	Next Unique Key Number	0	Date references found
H05	APPL	P0004A	User Defined Codes	6	Date references found
H05	APPL	P0004D	User Defined Code Alternate Descriptions	0	Date references found
88	APPL	P00050	Workstation Configuration	0	Date references found
H06	APPL	P00053	Host Configuration	0	Date references found
H05	APPL	P00055	User Defined Code Search and Select	0	Date references found
00	APPL	P0006	Business Units	4	Date references found
00	APPL	P00065	Translate Business Units	0	Date references found
09	APPL	P0006A	Organizational Structures	0	Date references found
00	APPL	P00066	Business Unit Search	0	Date references found
00	APPL	P00071	Work Day Calendar	77	Date references found
00	APPL	P00071W	Work Day Calendar Search & Select	0	Date references found
30	APPL	P000721	Work Day Calendar Transaction Revisions	6	Date references found
00	APPL	P0008	Fiscal Date Patterns	9	Date references found
00	APPL	P00055	Daylight Savings Rules	2	Date references found
00	APPL	P00008	Set 52 Period Dates	11	Date references found
00	APPL	P00091	Supplemental Data Setup	7	Date references found
00	APPL	P00092	Supplemental Data	36	Date references found
00	APPL	P00093	Flash Messages	3	Date references found
00	APPL	P0009M	CTI Interface	2	Date references found
00	APPL	P0010	Companies	106	Date references found

The other report shows the date reference details.

The following illustration is an example of an event rule date reference detail report.

R9000080ScanDetailReport(D98_09_01T1301) - Notepad

File Edit Search Help

EVENT RULES DATE REFERENCE SCAN REPORT(DETAILED)

Type	Object Name	Version	Form Name	CtrlId	Event	Line#
1	P0000		V0000B	43	Selection Changed	1
1	P0000		V0000B	1	Selection Changed	18
1	P0000		V0000B	1	Selection Changed	21
1	P0000		V0000D	0	Dialog is Initialized	6
1	P0000		V0000D	1	Selection Changed	37
1	P0002		V0002H	0	Write Grid Line-Before	2
1	P0002		V0002H	0	Write Grid Line-Before	3
1	P0002		V0002H	0	Write Grid Line-Before	4
1	P0002		V0002H	0	Write Grid Line-Before	6
1	P0002		V0002H	0	Write Grid Line-Before	7
1	P0002		V0002H	0	Write Grid Line-Before	11
1	P0002		V0002H	0	Write Grid Line-Before	12
1	P0002		V0002H	0	Write Grid Line-Before	14
1	P0002		V0002H	1	Row Exit & Changed - Asynch	12
1	P0002		V0002H	1	Row Exit & Changed - Asynch	19
1	P0002		V0002H	1	Row Exit & Changed - Asynch	44
1	P0002		V0002H	1	Row Exit & Changed - Asynch	53
1	P0002		V0002H	42	Col Exited & Changed - Asynch	1
1	P0002		V0002H	42	Col Exited & Changed - Asynch	2
1	P0002		V0002H	42	Col Exited & Changed - Asynch	4
1	P0002		V0002H	42	Col Exited & Changed - Asynch	5
1	P0002		V0002H	42	Col Exited & Changed - Asynch	7
1	P0002		V0002H	42	Col Exited & Changed - Asynch	8
1	P0002		V0002H	42	Col Exited & Changed - Asynch	9
1	P0002		V0002H	42	Col Exited & Changed - Asynch	10
1	P0002		V0002H	42	Col Exited & Changed - Asynch	12
1	P0002		V0002H	42	Col Exited & Changed - Asynch	13
1	P0004A		V0004AH	1	Update Grid Rec to DB-Before	3
1	P0004A		V0004AH	12	Selection Changed	1
1	P0004A		V0004AH	1	Add Grid Rec to DB - Before	3

t

Glossary

Glossary

AAI. See automatic accounting instruction.

action message. With OneWorld, users can receive messages (system-generated or user-generated) that have shortcuts to OneWorld forms, applications, and appropriate data. For example, if the general ledger post sends an action error message to a user, that user can access the journal entry (or entries) in error directly from the message. This is a central feature of the OneWorld workflow strategy. Action messages can originate either from OneWorld or from a third-party e-mail system.

ActiveX. A computing technology, based on object linking and embedding, that 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++.

alphanumeric character. A combination of letters, numbers, and symbols used to represent data. Contrast with numeric character and special character.

API. See application programming interface.

applet. A small application, such as a utility program or a limited-function spreadsheet. It is generally associated with the programming language Java, and in this context refers to Internet-enabled applications that can be passed from a Web browser residing on a workstation.

application. In the computer industry, the same as an executable file. In OneWorld, an interactive or batch application is a DLL that contains programming for a set of related forms that can be run from a menu to perform a business task such as Accounts Payable and Sales Order Processing. Also known as system.

application developer. A programmer who develops OneWorld applications using the OneWorld toolset.

application programming interface (API). A software function call that can be made from a program to access functionality provided by another program.

application workspace. The area on a workstation display in which all related forms within an application appear.

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 accounting instruction (AAI). A code that refers to an account in the chart of accounts. AAIs define rules for programs that automatically generate journal entries, including interfaces between Accounts Payable, Accounts Receivable, Financial Reporting, General Accounting systems. Each system that interfaces with the General Accounting system has AAIs. For example, AAIs can direct the General Ledger Post program to post a debit to a specific expense account and a credit to a specific accounts payable account.

batch header. The information that identifies and controls a batch of transactions or records.

batch job. A task or group of tasks you submit for processing that the system treats as a single unit during processing, for example, printing reports and purging files. The computer system performs a batch job with little or no user interaction.

batch processing. A method by which the system selects jobs from the job queue, processes them, and sends output to the outqueue. Contrast with interactive processing.

batch server. A server on which OneWorld batch processing requests (also called UBEs) are run instead of on a client, an application server, or an enterprise server. A batch server typically does not contain a database nor does it run interactive applications.

batch type. A code assigned to a batch job that designates to which J.D. Edwards system the associated transactions pertain, thus controlling which records are selected for processing. For

example, the Post General Journal program selects for posting only unposted transaction batches with a batch type of O.

batch-of-one immediate. A transaction method that allows a client application to perform work on a client workstation, 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.

BDA. See Business View Design Aid.

binary string (BSTR). A length prefixed string used by OLE automation data manipulation functions. Binary Strings are wide, double-byte (Unicode) strings on 32-bit Windows platforms.

Boolean Logic Operand. In J.D. Edwards reporting programs, the parameter of the Relationship field. The Boolean logic operand instructs the system to compare certain records or parameters. Available options are:

EQ	Equal To.
LT	Less Than.
LE	Less Than or Equal To.
GT	Greater Than.
GE	Greater Than or Equal To.
NE	Not Equal To.
NL	Not Less Than.
NG	Not Greater Than.

browser. A client application that translates information sent by the World Wide Web. A client must use a browser to receive, manipulate, and display World Wide Web information on the desktop. Also known as a Web browser.

BSTR. See binary string.

business function. An encapsulated set of business rules and logic that can normally be reused 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-OneWorld 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 created using through event rules rather than C programming. Contrast with embedded event rule. See also event rule.

business view. Used by OneWorld applications to access data from database tables. A business view is a means for selecting specific columns from one or more tables whose data 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). A OneWorld GUI tool for creating, modifying, copying, and printing business views. The tool uses a graphical user interface.

category code. In user defined codes, a temporary title for an undefined category. For example, if you are adding a code that designates different sales regions, you could change category code 4 to Sales Region, and define E (East), W (West), N (North), and S (South) as the valid codes. Sometimes referred to as reporting codes.

central objects. Objects that reside in a central location and consist of two parts: the central objects data source and central C components. The central objects data source contains OneWorld specifications, which are stored in a relational database. Central C components contain business function source, header, object, library, and DLL files and are usually stored in directories on the deployment server. Together they make up central objects.

check-in location. The directory structure location for the package and its set of replicated objects. This is usually
\\deploymentserver\release\path_code\package\packagename. The sub-directories under this path are where the central C components (source, include, object, library, and DLL file) for business functions are stored.

child. See parent/child form.

client/server. A relationship between processes running on separate machines. The server process is a provider of software services. The client is a consumer of those services. In essence, client/server provides a clean separation of function based on the idea of service. A server can service many clients at the same time and regulate their access to shared resources. There is a many-to-one relationship between clients and a server,

respectively. Clients always initiate the dialog by requesting a service. Servers passively wait for requests from clients.

CNC. See configurable network computing.

configurable client engine. Allows user flexibility at the interface level. Users can easily move columns, set tabs for different data views, and size grids according to their needs. The configurable client engine also enables the incorporation of Web browsers in addition to the Windows 95- and Windows NT-based interfaces.

configurable network computing. An application architecture that allows interactive and batch applications, composed of a single code base, to run across a TCP/IP network of multiple server platforms and SQL databases. The applications consist of reusable 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.

constants. Parameters or codes that you set and the system uses to standardize information processing by associated programs. Some examples of constants are: validating bills of material online and including fixed labor overhead in costing.

control. Any data entry point allowing the user to interact with an application. For example, check boxes, pull-down lists, hyper-buttons, entry fields, and similar features are controls.

core. The central and foundation systems of J.D. Edwards software, including General Accounting, Accounts Payable, Accounts Receivable, Address Book, Financial Reporting, Financial Modeling and Allocations, and Back Office.

custom gridlines. A grid row that does not come from the database, for example, totals. To display a total in a grid, sum the values and insert a custom gridline to display the total. Use the system function Insert Grid Row Buffer to accomplish this.

data dictionary. The OneWorld method for storing and managing data item definitions and specifications. J.D. Edwards has an active data dictionary, which means it is accessed at runtime.

data mart. Department-level decision support databases. They usually draw their data from an enterprise data warehouse that serves as a source of

consolidated and reconciled data from around the organization. Data marts can be either relational or multidimensional databases.

data replication. In a replicated environment, multiple copies of data are maintained on multiple machines. There must be a single source that “owns” the data. This ensures that the latest copy of data can be applied to a primary place and then replicated as appropriate. This is in contrast to a simple copying of data, where the copy is not maintained from a central location, but exists independently of the source.

data source. A specific instance of a database management system running on a computer. Data source management is accomplished through Object Configuration Manager (OCM) and Object Map (OM).

data structure. A group of data items that can be used for passing information between objects, for example, between two forms, between forms and business functions, or between reports and business functions.

data warehouse. A database used for reconciling and consolidating data from multiple databases before it is distributed to data marts for department-level decision support queries and reports. The data warehouse is generally a large relational database residing on a dedicated server between operational databases and the data marts.

data warehousing. Essentially, data warehousing involves off-loading operational data sources to target databases that will be used exclusively for decision support (reports and queries). There are a range of decision support environments, including duplicated database, enhanced analysis databases, and enterprise data warehouses.

database. A continuously updated collection of all information a system uses and stores. Databases make it possible to create, store, index, and cross-reference information online.

database driver. Software that connects an application to a specific database management system.

database server. A server that stores data. A database server does not have OneWorld logic.

DCE. See distributed computing environment.

default. A code, number, or parameter value that is assumed when none is specified.

detail. The specific pieces of information and data that make up a record or transaction. Contrast with summary.

detail area. A control that is found in OneWorld applications and functions similarly to a spreadsheet grid for viewing, adding, or updating many rows of data at one time.

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.

distributed computing environment (DCE). A set of integrated software services that allows software running on multiple computers to perform in a manner that is seamless and transparent to the end-users. DCE provides security, directory, time, remote procedure calls, and files across computers running on a network.

DLL. See dynamic link library.

duplicated database. A decision support database that contains a straightforward copy of operational data. The advantages involve improved performance for both operational and reporting environments. See also enhanced analysis database, enterprise data warehouse.

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.

embedded event rule. An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with business function event rule. See also event rule.

employee work center. This is a central location for sending and receiving all OneWorld 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 OneWorld messages or Microsoft Exchange.

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.

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 data warehouse. A complex solution that involves data from many areas of the enterprise. This environment requires a large relational database (the data warehouse) that is a central repository of enterprise data, which is clean, reconciled, and consolidated. From this repository, data marts retrieve data to provide department-level decisions. See also duplicated database, enhanced analysis database.

enterprise server. A database server and logic server. See database server. Also referred to as host.

ERP. See enterprise resource planning.

event. An action that occurs when an interactive or batch application is running. Example events are tabbing out of an edit control, clicking a push button, initializing a form, or performing a page break on a report. The GUI operating system uses miniprograms to manage user activities within a form. Additional logic can be attached to these miniprograms and used to give greater functionality to any event within a OneWorld application or report using event rules.

event rule. Used to create complex business logic without the difficult syntax that comes with many programming languages. These logic statements can be attached to applications or database events and are executed when the defined event occurs, such as entering a form, selecting a menu bar option, page breaking on a report, or selecting a record. An event rule can validate data, send a message to a user, call a business function, as well as many other actions. There are two types of event rules:

- 1 Embedded event rules.

2 Business function event rules.

executable file. A computer program that can be run from the computer's operating system. Equivalent terms are "application" and "program."

exit. 1) To interrupt or leave a computer program by pressing a specific key or a sequence of keys. 2) An option or function key displayed on a form that allows you to access another form.

facility. 1) A separate 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. 2) In Home Builder and ECS, a facility is a collection of computer language statements or programs that provide a specialized function throughout a system or throughout all integrated systems. For example, DREAM Writer and FASTR are facilities.

FDA. See Form Design Aid.

find/browse. A type of form used to:

- 1 Search, view, and select multiple records in a detail area.
- 2 Delete records.
- 3 Exit to another form.
- 4 Serve as an entry point for most applications.

firewall. A set of technologies that allows an enterprise to test, filter, and route all incoming messages. Firewalls are used to keep an enterprise secure.

fix/inspect. A type of form used to view, add, or modify existing records. A fix/inspect form has no detail area.

form. An element of OneWorld's graphical user interface that contains controls by which a user can interact with an application. Forms allow the user to input, select, and view information. A OneWorld application might contain multiple forms. In Microsoft Windows terminology, a form is known as a dialog box.

Form Design Aid (FDA). The OneWorld GUI development tool for building interactive applications and forms.

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 OneWorld:

- 1 Find/browse.
- 2 Fix/inspect.
- 3 Header detail.
- 4 Headerless detail.
- 5 Message.
- 6 Parent/child.
- 7 Search/select.

fourth generation language (4GL). A programming language that focuses on what you need to do and then determines how to do it. Structured Query Language is an example of a 4GL.

graphical user interface (GUI). A computer interface that is graphically based as opposed to being character-based. An example of a character-based interface is that of the AS/400. An example of a GUI is Microsoft Windows. Graphically based interfaces allow pictures and other graphic images to be used in order to give people clues on how to operate the computer.

grid. See detail area.

GUI. See graphical user interface.

header. Information at the beginning of a table or form. This information is used to identify or provide control information for the group of records that follows.

header/detail. A type of form used to add, modify, or delete records from two different tables. The tables usually have a parent/child relationship.

headerless detail. A type of form used to work with multiple records in a detail area. The detail area is capable of receiving input.

hidden selections. Menu selections you cannot see until you enter HS in a menu's Selection field. Although you cannot see these selections, they are available from any menu. They include such items as Display Submitted Jobs (33), Display User Job Queue (42), and Display User Print Queue (43). The Hidden Selections window displays three categories of selections: user tools, operator tools, and programmer tools.

host. In the centralized computer model, a large timesharing computer system that terminals communicate with and rely on for processing. In contrast with client/server in that those users work at computers that perform much of their own processing and access servers that provide services

such as file management, security, and printer management.

HTML. See hypertext markup language.

hypertext markup language. A markup language used to specify the logical structure of a document rather than the physical layout. Specifying logical structure makes any HTML document platform independent. You can view an HTML document on any desktop capable of supporting a browser. HTML can include active links to other HTML documents anywhere on the Internet or on intranet sites.

index. Represents both an ordering of values and a uniqueness of values that provide efficient access to data in rows of a table. An index is made up of one or more columns in the table.

inheritance. The ability of a class to receive all or parts of the data and procedure definitions from a parent class. Inheritance enhances development through the reuse of classes and their related code.

install system code. See system code.

integrated toolset. Unique to OneWorld is an industrial-strength toolset embedded in the already comprehensive business applications. This toolset is the same toolset used by J.D. Edwards to build OneWorld interactive and batch applications. Much more than a development environment, however, the OneWorld integrated toolset handles reporting and other batch processes, change management, and basic data warehousing facilities.

interactive processing. Processing actions that occur in response to commands you enter directly into the system. During interactive processing, you are in direct communication with the system, and it might prompt you for additional information while processing your request. See also online. Contrast with batch processing.

interface. A link between two or more computer systems that allows these systems to send information to and receive information from one another.

Internet. The worldwide constellation of servers, applications, and information available to a desktop client through a phone line or other type of remote access.

interoperability. The ability of different computer systems, networks, operating systems, and

applications to work together and share information.

intranet. A small version of the Internet usually confined to one company or organization. An intranet uses the functionality of the Internet and places it at the disposal of a single enterprise.

IP. A connection-less communication protocol that by itself provides a datagram service. Datagrams are self-contained packets of information that are forwarded by routers based on their address and the routing table information contained in the routers. Every node on a TCP/IP network requires an address that identifies both a network and a local host or node on the network. In most cases the network administrator sets up these addresses when installing new workstations. In some cases, however, it is possible for a workstation, when booting up, to query a server for a dynamically assigned address.

IServer Service. Developed by J.D. Edwards, 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.

ISO 9000. A series of standards established by the International Organization for Standardization, designed as a measure of product and service quality.

J.D. Edwards Database. See JDEBASE Database Middleware.

Java. An Internet executable language that, like C, is designed to be highly portable across platforms. This programming language was developed by Sun Microsystems. Applets, or Java applications, can be accessed from a web browser and executed at the client, provided that the operating system or browser is Java-enabled. (Java is often described as a scaled-down C++). Java applications are platform independent.

Java Database Connectivity (JDBC). The standard way to access Java databases, set by Sun Microsystems. This standard allows you to use any JDBC driver database.

jde.ini. J.D. Edwards file (or member for AS/400) that provides the runtime settings required for OneWorld initialization. Specific versions of the file/member must reside on every machine running OneWorld. This includes workstations and servers.

JDEBASE Database Middleware. J.D. Edwards 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 datasource 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 OneWorld 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 used by business functions to invoke other business functions.

JDENET. J.D. Edwards proprietary middleware software. JDENET is a messaging software package.

JDENET communications middleware. J.D. Edwards proprietary communications middleware package for OneWorld. 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 OneWorld supported platforms.

job queue. A group of jobs waiting to be batch processed. See also batch processing.

just in time installation (JITI). OneWorld's method of dynamically replicating objects from the central object location to a workstation.

just in time replication (JITR). OneWorld's method of replicating data to individual workstations. OneWorld replicates new records (inserts) only at the time the user needs the data. Changes, deletes, and updates must be replicated using Pull Replication.

KEY. A column or combination of columns that identify one or more records in a database table.

leading zeros. A series of zeros that certain facilities in J.D. Edwards systems place in front of a value you enter. This 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 you enter. The result appears as: 00004567.

level of detail. 1) The degree of difficulty of a menu in J.D. Edwards software. The levels of detail for menus are as follows:

A	Major Product Directories.
B	Product Groups.
1	Basic Operations.
2	Intermediate Operations.
3	Advanced Operations.
4	Computer Operations.
5	Programmers.
6	Advanced Programmers Also known as menu levels.

2) The degree to which account information in the General Accounting system is summarized. The highest level of detail is 1 (least detailed) and the lowest level of detail is 9 (most detailed).

MAPI. See Messaging Application Programming Interface.

master table. A database table 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.

menu. A menu that displays numbered selections. Each of these selections represents a program or another menu. To access a selection from a menu, type the selection number and then press Enter.

menu levels. See level of detail.

menu masking. A security feature of J.D. Edwards systems that lets you prevent individual users from accessing specified menus or menu selections. The system does not display the menus or menu selections to unauthorized users.

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.

middleware. A general term that covers all the distributed software needed to support interactions between clients and servers. Think of it as the software that's in the middle of the client/server

system or the “glue” that lets the client obtain a service from a server.

modal. A restrictive or limiting interaction 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.

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.

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.

network computer. As opposed to the personal computer, the network computer offers (in theory) lower cost of purchase and ownership and less complexity. Basically, it is a scaled-down PC (very little memory or disk space) that can be used to access network-based applications (Java applets, ActiveX controls) via a network browser.

network computing. Often referred to as the next phase of computing after client/server. While its exact definition remains obscure, it generally encompasses issues such as transparent access to computing resources, browser-style front-ends, platform independence, and other similar concepts.

next numbers. A feature you use to control the automatic numbering of such items as new G/L accounts, vouchers, and addresses. It lets you specify a numbering system and provides a method to increment numbers to reduce transposition and typing errors.

numeric character. Digits 0 through 9 that are used to represent data. Contrast with alphanumeric characters.

object. A self-sufficient entity that contains data as well as the structures and functions used to manipulate the data. For OneWorld purposes, an object is a reusable entity that is based on software specifications created by the OneWorld toolset. See also object librarian.

object configuration manager (OCM).

OneWorld’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 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 reusable in building applications. It provides check-out and check-in capabilities for developers, and it controls the creation, modification, and use of OneWorld Objects. The Object Librarian supports multiple environments (such as production and development) and allows objects to be easily moved from one environment to another.

object linking. When an object is linked to another document, a reference is created with the file the object is stored in, as well as with the application that created it. When the object is modified, either from the compound document or directly through the file it is saved in, 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 way to integrate objects from diverse applications, such as graphics, charts, spreadsheets, text, or an audio clip from a sound program. See also object embedding, object linking.

object-based technology (OBT). A technology that supports some of the main principles of object-oriented technology: classes, polymorphism, inheritance, or encapsulation.

object-oriented technology (OOT). Brings software development past procedural programming into a world of reusable programming that simplifies development of applications. Object orientation is based on the following principles: classes, polymorphism, inheritance, and encapsulation.

OCM. See object configuration manager.

ODBC. See open database connectivity.

OLE. See object linking and embedding.

OneWorld. A combined suite of comprehensive, mission-critical business applications and an embedded toolset for configuring those applications to unique business and technology requirements. OneWorld is built on the Configurable Network Computing technology- J.D. Edwards' own application architecture, which extends client/server functionality to new levels of configurability, adaptability, and stability.

OneWorld application. Interactive or batch processes that execute the business functionality of OneWorld. They consist of reusable business functions and associated data that are platform independent and can be dynamically configured across a TCP/IP network.

OneWorld object. A reusable 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.

OneWorld process. Allows OneWorld clients and servers to handle processing requests and execute transactions. A client runs one process, and servers can have multiple instances. OneWorld processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

OneWorld Web development computer. A standard OneWorld Windows developer computer with the additional components installed:

- JFC (0.5.1).
- Generator Package with Generator.Java and JDECOM.dll.
- R2 with interpretive and application controls/form.

online. Computer functions over which the system has continuous control. Users are online with the system when working with J.D. Edwards system provided forms.

open database connectivity (ODBC). Defines a standard interface for different technologies to process data between applications and different data sources. The ODBC interface is made up of a 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.

operand. See Boolean Logic Operand.

output. Information that the computer transfers from internal storage to an external device, such as a printer or a computer form.

output queue. See print queue.

package. OneWorld 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 on the deployment server the install program can find them. It is a point-in-time "snap shot" of the central objects on the deployment server.

package location. The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\ package name. The sub-directories under this path are where the replicated objects for the package will be placed. This is also referred to as where the package is built or stored.

parameter. A number, code, or character string you specify in association with a command or program. The computer uses parameters as additional input or to control the actions of the command or program.

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.

partitioning. A technique for distributing data to local and remote sites to place data closer to the users who access. Portions of data can be copied to different database management systems.

path code. A pointer to a specific set of objects. A path code is used to locate:

1. Central Objects.

2. Replicated Objects.

platform independence. A benefit of open systems and Configurable Network Computing.

Applications that are composed of a single code base can be run across a TCP/IP network consisting of various server platforms and SQL databases.

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.

portability. Allows the same application to run on different operating systems and hardware platforms.

primary key. A column or combination of columns that uniquely identifies each row in a table.

print queue. A list of tables, such as reports, that you have submitted to be written to an output device, such as a printer. The computer spools the tables until it writes them. After the computer writes the table, the system removes the table identifier from the list.

processing option. A feature of the J.D. Edwards reporting system that allows you to supply parameters to direct the functions of a program. For example, processing options allow you to specify defaults for certain form displays, control the format in which information prints on reports, change how a form displays information, and enter beginning dates.

program temporary fix (PTF). A representation of changes to J.D. Edwards software that your organization receives on magnetic tapes or diskettes.

published table. Also called a “Master” table, this is the central copy to be replicated to other machines. 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 OneWorld methods for replicating data to individual workstations. Such machines are set up as Pull Subscribers using OneWorld’s data replication tools. The only time 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).

purge. The process of removing records or data from a system table.

query by example (QBE). Located at the top of a detail area, it is used to search for data to be displayed in the detail area.

redundancy. Storing exact copies of data in multiple databases.

regenerable. Source code for OneWorld 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.

relationship. Links tables together and facilitates joining business views for use in an application or report. Relationships are created based on indexes.

release/release update. A “release” contains major new functionality, and a “release update” contains an accumulation of fixes and performance enhancements, but no new functionality.

replicated object. A copy or replicated set of the central objects must reside on each client and server that run OneWorld. The path code indicates the directory the directory where these objects are located.

run. To cause the computer system to perform a routine, process a batch of transactions, or carry out computer program instructions.

SAR. See software action request.

scalability. Allows software, architecture, network, or hardware growth that will support software as it grows in size or resource requirements. The ability to reach higher levels of performance by adding microprocessors.

search/select. A type of form used to search for a value and return it to the calling field.

selection. Found on J.D. Edwards 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.

server. Provides the essential functions for furnishing services to network users (or clients) and provides management functions for network administrators. Some of these functions are storage of user programs and data and management functions for the file systems. It may not be possible for one server to support all users with the required services. Some examples of dedicated servers that handle specific tasks are backup and archive servers, application and database servers.

servlet. Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions. Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are to the server-side what applets are to the client-side.

software. The operating system and application programs that tell the computer how and what tasks to perform.

software action request (SAR). An entry in the AS/400 database used for requesting modifications to J.D. Edwards software.

special character. A symbol used to represent data. Some examples are *, &, #, and /. Contrast with alphanumeric character and numeric character.

specifications. A complete description of a OneWorld object. Each object has its own specification, or name, which is used to build applications.

spool. The function by which the system stores generated output to await printing and processing.

spooled table. A holding file for output data waiting to be printed or input data waiting to be processed.

SQL. See structured query language.

static text. Short, descriptive text that appears next to a control variable or field. When the variable or field is enabled, the static text is black; when the variable or field is disabled, the static text is gray.

store and forward. A transaction method that allows a client application to perform work and, at a later time, complete that work by connecting to a server application. This often involves uploading data residing on a client to a server.

structured query language (SQL). A fourth generation language used as an industry standard

for relational database access. It can be used to create databases and to retrieve, add, modify, or delete data from databases. SQL is not a complete programming language because it does not contain control flow logic.

subfile. See detail.

submit. See run.

subscriber. The server that is responsible for the replicated copy of a Published Table. Such servers are identified in the Subscriber Table.

subscriber table. The Subscriber Table (F98DRSUB), which is stored on the Publisher Server with the Data Replication Publisher Table (F98DRPUB) identifies all of the Subscriber machines for each Published Table.

subsystem job. Within OneWorld, subsystem jobs are batch processes that continually run independently of, but asynchronously with, OneWorld applications.

summary. The presentation of data or information in a cumulative or totaled manner in which most of the details have been removed. Many of the J.D. Edwards systems offer forms and reports that are summaries of the information stored in certain tables. Contrast with detail.

system. See application.

System Code. System codes are a numerical representation of J.D. Edwards and customer systems. For example, 01 is the system code for Address Book. System codes 55 through 59 are reserved for customer development by customers. Use system codes to categorize within OneWorld. For example, when establishing user defined codes (UDCs), you must include the system code the best categorizes it. When naming objects such as applications, tables, and menus, the second and third characters in the object's name is the system code for that object. For example, G04 is the main menu for Accounts Payable, and 04 is its system code.

system function. A program module, provided by OneWorld, available to applications and reports for further processing.

table. A two-dimensional entity made up of rows and columns. All physical data in a database are stored in tables. A row in a table contains a record of related information. An example would be a record in an Employee table containing the Name,

Address, Phone Number, Age, and Salary of an employee. Name is an example of a column in the employee table.

table design aid (TDA). A OneWorld 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 a OneWorld database trigger, you must first determine which event will activate the trigger. Then, use Event Rules Design to create the trigger. Although OneWorld allows event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

TCP/IP. Transmission Control Protocol/Internet Protocol. The original TCP protocol was developed as a way to interconnect networks using many different types of transmission methods. TCP provides a way to establish a connection between end systems for the reliable delivery of messages and data.

TCP/IP services port. Used by a particular server application to provide whatever service the server is designed to provide. The port number must be readily known so that an application programmer can request it by name.

TDA. See table design aid.

Terminal Identification. The workstation ID number. Terminal number of a specific terminal or IBM user ID of a particular person for whom this is a valid profile. Header Field: Use the Skip to Terminal/User ID field in the upper portion of the form as an inquiry field in which you can enter the number of a terminal or the IBM user ID of a specific person whose profile you want the system to display at the top of the list. When you first access this form, the system automatically enters the user ID of the person signed on to the system. Detail Field: The Terminal/User ID field in the lower portion of the form contains the user ID of the person whose profile appears on the same line. A code identifying the user or terminal for which you accessed this window.

third generation language (3GL). A programming language that requires detailed information about

how to complete a task. Examples of 3GLs are COBOL, C, Pascal and FORTRAN.

trigger. Allow 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 associated with the data item. OneWorld also has three visual assist triggers: calculator, calendar and search form.

UDC Edit Control. Use a User-Defined Code (UDC) Edit Control for a field that accepts only specific values defined in a UDC table. Associate a UDC edit control with a database item or dictionary item. The visual assist Flashlight automatically appears adjacent to the UDC edit control field. When you click on the visual assist Flashlight, the attached search and select form displays valid values for the field. To create a UDC Edit Control, you must:

- Associate the data item with a specific UDC table in the Data Dictionary.
- Create a search and select form for displaying valid values from the UDC table.

uniform resource locator (URL). Names the address of a document on the Internet or an intranet. The following is an example of URL: <http://www.jdedwards.com>. This is J.D. Edwards Internet address.

user defined code (type). The identifier for a table of codes with a meaning you define for the system, such as ST for the Search Type codes table in Address Book. J.D. Edwards systems provide a number of these tables and allow you to create and define tables of your own. User defined codes were formerly known as descriptive titles.

user defined codes (UDC). Codes within software that users can define, relate to code descriptions, and assign valid values. Sometimes user defined codes are referred to as a generic code table. Examples of such codes are unit-of-measure codes, state names, and employee type codes.

valid codes. The allowed codes, amounts, or types of data that you can enter in a field. The system verifies the information you enter against the list of valid codes.

visual assist. Forms that can be invoked from a control to assist the user in determining what data belongs in the control.

vocabulary overrides. A feature you can use to override field, row, or column title text on forms and reports.

wchar_t. Internal type of a wide character. Used for writing portable programs for international markets.

web client. Any workstation that contains an internet browser. The web client communicates with the web server for OneWorld data.

web server. Any workstation that contains the IServer service, SQL server, Java menus and applications, and Internet middleware. The web server receives data from the web client, and passes the request to the enterprise server. When the enterprise server processes the information, it sends it back to the web server, and the web server sends it back to the web client.

window. See form.

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 remote database server usually containing subsets of data replicated from a master database server. This server does not performance an application or batch processing. It may or may not have OneWorld running (in order to replicate data).

worldwide web. A part of the Internet that can transmit text, graphics, audio, and video. The World Wide Web allows clients to launch local or remote applications.

z file. For store and forward (network disconnected) user, OneWorld 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, OneWorld 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 will then update the records in your transaction files.

Index

Index

A

- Access, cache, using an index, 13–9
- Access32, index limitations for various databases, performance tips, 25–6
- Action, message, 22–5
- Action code standards, cache, 13–37
 - CACHE_ADD, 13–37
 - CACHE_ADD_UPDATE, 13–38
 - CACHE_DELETE, 13–38
 - CACHE_GET, 13–37
 - CACHE_UPDATE, 13–38
 - CACHE_CLOSE_CURSOR, 13–39
 - CACHE_GET_NEXT, 13–39
 - CACHE_TERMINATE, 13–39
 - CACHE_TERMINATE_ALL, 13–39
- Active message, defining, 21–29
- Add, menu/toolbar items
 - Find/Browse form, B–6
 - Header Detail form, B–26
 - Headerless Detail form, B–33
 - Parent/Child form, B–13
 - Search/Select form, B–38
- Add button, processing, 11–27
- Add entry row to grid
 - Header Detail form, B–22
 - Headerless Detail form, B–29
- Add Object Form, 12–42
- Add Object form, 2–8
- Adding
 - an object, 2–8
 - API record, subsystem table, 20–7
 - application, J.D. Edwards naming standards, 9–6, 10–20
 - bitmap strip, 10–36
 - business function, object codes, 12–35
 - business functions
 - codes, 12–36
 - related functions, 12–38
 - related tables and functions, 12–36
 - business view, 5–3
 - J.D. Edwards naming standards, 5–4
 - comment lines to event rules, 11–39
 - glossary for languages, 3–41
 - interactive application, 9–5
 - interactive message data item, 21–15, 21–17
 - language specific media object attachment, 10–122
 - processing option, J.D. Edwards naming standards, 14–10
 - record, subsystem table, 20–8
 - tables, 4–3
 - J.D. Edwards naming standards, 4–4
- Advanced functions, expression manager, 11–93
 - date functions, 11–93
 - general functions, 11–94
 - text functions, 11–94
 - trig functions, 11–94
- Advancing APIs
 - cursor, 13–17
 - non-cursor, 13–18
- Advantages, currency, 16–1
- Alias, used in an RPG program, 3–13
- Aligning
 - alignment grid to align controls, 10–30
 - controls, 10–29
 - group, 10–29
- All grid recs deleted from DB, runtime processing, 11–30
- Allocates, OneWorld, memory for sections, 18–13
- API record, subsystem table, adding, 20–7
- API set, JDECACHE, 13–3
- APIs, calling Work Center, 22–20
- Application
 - adding, J.D. Edwards naming standards, 9–6, 10–20
 - batch processing, 18–1
 - event rules, 11–4
 - filed to highlight, error code, 21–6
 - Math_Numeric currency, data structure, 16–5
 - Message, 21–1
- Application Programming Interfaces, 12–23
- Application specific parameters
 - begin document, 12–70

- edit document, 12–73
- end document, 12–74
- Application text modification rules, A–6
- Applications
 - add to a menu, 17–13
 - changing row and column text, 3–21
 - debugging, 23–11
 - errors, encountering, 23–26
 - generating
 - all, 24–15
 - one, 24–14
 - interactive, adding, 9–5
 - naming conventions, 2–14
 - performance, 25–1
 - performance tips, things to look for, 25–3
 - understanding, 1–4, 9–3
 - understanding how to build, 1–5
- Assign, value, 11–89
- Assignment, creating, 11–89
 - expression, 11–91
- Assignment display, expression manager, 11–92
- Associating
 - an item or description with a control, 10–92
 - database items, 10–91
 - database items, dictionary items, or descriptions, 10–91
 - description or item with a control, 10–92
 - descriptions, 10–91
 - dictionary items, 10–91
- Asynchronous, business functions, 11–118
- Asynchronous events, how threaded events work, 11–116
- Asynchronous processing, working with, 11–115
- Attaching
 - business function to an event, 11–69
 - data structure template, message, 21–20
 - default triggers, 3–23
 - default value triggers, 3–25
 - display rule triggers, 3–31
 - edit rule triggers, 3–28
 - event rules, Menu/Toolbar Exit, 10–38
 - functions, 11–61
 - interactive message, data item, 21–21
 - logic, custom section, 18–16
 - logic to joined sections, 18–12
 - next number triggers, 3–36
 - processing options, template, 14–17
 - smart field triggers, 3–38

- system function to an event, 11–61
- user defined codes triggers, 3–33
- visual assist triggers, 3–26
- Attachment, adding, language specific media object, 10–122
- Automatic error, error setting, 21–7
- Automatic Line Numbering, 11–59
- Available batch objects, processing, 18–34
- Available business functions, business function builder form, 12–46
- Available information, expression manager, 11–93
- Available Objects, 18–33
- Available objects, processing, 11–12
- Available objects and runtime data structures, 11–9
- Available operations, Table I/O, 11–97

B

- Balance table closes and opens, business function performance, performance tips, 25–15
- Balance table opens and closes, business function performance, performance tips, 25–15
- Batch application event rules variables, 11–54
- Batch application performance, performance tips, 25–12
 - setting up lever breaks, 25–12
- Batch Applications, searching, 7–6
- Batch applications, business functions, debugging, 23–18
- Batch error message, 22–1
- Batch error messages, understanding, 22–3
- Batch events
 - Do_Section event, 18–23
 - hide object, 18–32
 - set sequence append flag, 18–31
 - stop section processing, 18–32
 - system functions within, 18–31
 - understanding, 18–23
- Batch processes, OneWorld, subsystem jobs, 20–1
- Batch Processing, understanding, 18–3
- Batch processing, application, 18–1
- Batch Runtime, processing, 18–33
- Batch section processing, understanding, 18–9
- Batch sections, understanding level-break processing, 18–19

- BC assigned database values, runtime processing, 11–19
- Begin DLL section, build output, 12–52
- Begin document
 - application specific parameters, 12–70
 - common parameters, 12–68, 12–71
 - function name, 12–67, 12–70
 - hook up tips, 12–68
 - naming, transaction master business function modules, 12–67
 - special logic/processing required, 12–68, 12–70
 - typical users and hook up, 12–71
 - what does it do, 12–67
- Bitmap controls, creating, 10–66
- Bitmap strip, adding, 10–36
- Bitmap Strips, Adding, 10–36
- Breakpoint Manager, Event Rules Debugger, 23–9
- Breakpoints, debugging, understanding, 23–9
- BrowsER, 11–121
 - expanded node, 11–125
 - filter ER records, 11–126
 - options, working with, 11–124
 - search, 11–126
 - show object ID, 11–125
 - working with, 11–121
- Build
 - applications, 1–5
 - triggers option, 16–3
- Build all, performing, 12–54
- Build options, setting, 12–49
- Build output
 - begin DLL section, 12–52
 - business function builder form, 12–46
 - compile section, 12–52
 - link section, 12–53
 - makefile section, 12–52
 - reading, 12–51
 - rebase section, 12–53
 - summary section, 12–54
- Building
 - single business function, 12–47
 - transaction master business functions, 12–78
- Business function
 - creating, documentation, 12–87, 12–88
 - designing, 2–29
 - documentation, 12–87
 - generating, 12–95
 - template, 12–90
 - viewing, 12–98
 - event, attaching, 11–69
 - Event Rules, OneWorld tools, 1–11
 - event rules, 11–3
 - generating, documentation, 12–95
 - level-break, creating, 22–15
 - naming conventions, 2–16
 - viewing, documentation, 12–98
- Business function - values to pass, documentation, viewing, 12–99
- Business function API, manual error, setting, 21–8
- Business Function Builder, DLLs, 12–4
- Business function builder
 - resolving errors, 12–62
 - working with, 12–45
- Business function builder form
 - available business functions, 12–46
 - Build output, 12–46
 - functions list, 12–46
 - understanding, 12–45
- Business function data structure, creating, 6–12
 - level-break, 22–13
- Business function data structures
 - naming conventions, 2–16
 - user generated, 6–2
- Business Function Design, OneWorld tools, 1–10
- Business Function Design form, 12–44
- Business function document viewer, documentation, viewing, 12–100
- Business function event rules
 - creating, 12–7
 - creating or editing event rules, 12–11
- Business function modification rules, A–9
- Business Function Object Codes, adding, 12–35
- Business function objects, linking, 12–49
- Business function performance, performance tips, 25–14
 - balance table closes and opens, 25–15
 - balance table opens and closes, 25–15
 - memory allocation, 25–14
- Business function search, documentation, viewing, 12–98
- Business function source files, structure, 12–20
- Business function source librarian, working with, 2–29
- Business Functions
 - building all, 12–54

- searching, 7–7
- Business functions
 - adding
 - codes, 12–36
 - object codes, 12–35
 - related functions, 12–38
 - related tables and functions, 12–36
 - asynchronous, 11–118
 - attached to batch applications, debugging, 23–18
 - attached to interactive applications, debugging, 23–15
 - building a single, 12–47
 - changing, parent DLL, 12–39
 - components, 12–3
 - debugging, 12–38
 - Visual C++, 23–15
 - description standards, cache business function source description, 13–35
 - displaying a reference, 2–28
 - features, 12–2
 - OneWorld tools, 12–1
 - standard naming convention, cache business function source name, 13–35
 - working with, 12–35
- Business functions or forms, multiple, using the same cache, 13–13
- Business view
 - adding, 5–3
 - J.D. Edwards naming standards, 5–4
 - choosing a table, 5–8
 - choosing data items, 5–9
 - creating a table join, 5–14
 - creating a table union, 5–16
 - generating, 5–16
 - using select distinct, 5–10
- Business View Design
 - OneWorld tools, 1–9, 5–1
 - working with, 5–7
- Business view modification rules, A–7
- Business view performance, performance tips, 25–9
 - joined business view versus table I/O, 25–10
 - restrict the number of fields, 25–9
 - single-table or multiple-table, 25–9
 - table I/O versus joined business view, 25–10
 - unions, 25–9
- Business Views, searching, 7–8

- Business views
 - naming conventions, 2–13
 - selecting, 10–20
- Button clicked, runtime processing, 11–26
- Buttons, Message form, B–39

C

- C business functions
 - multi-level error messaging, 21–11
 - understanding, 12–15
- C++, business functions, debugging, 23–15
- Cache
 - action code standards, 13–37
 - CACHE_ADD, 13–37
 - CACHE_ADD_UPDATE, 13–38
 - CACHE_CLOSE_CURSOR, 13–39
 - CACHE_DELETE, 13–38
 - CACHE_GET, 13–37
 - CACHE_GET_NEXT, 13–39
 - CACHE_TERMINATE, 13–39
 - CACHE_TERMINATE_ALL, 13–39
 - CACHE_UPDATE, 13–38
 - clearing versus terminating, 13–36
 - initializing, 13–7
 - JDECACHE management APIs, manage, 13–3
 - name, terminating versus clearing cache, 13–36
 - OneWorld, 13–1
 - programming standards, 13–36
 - general, 13–36
 - same in multiple business functions or forms, 13–13
 - structure, 12–75
 - detail, 12–77
 - header, 12–76
 - terminating versus clearing, 13–36
 - using an index to access, 13–9
- Cache business function description, source file, function name, 13–35
- Cache business function source description, business functions, description standards, 13–35
- Cache business function source name, standard naming convention, business functions, 13–35

- Cache business functions, data items, data structure, 13–37
- Cache data structure, defining, 13–37
- Cache name, terminating versus clearing cache, 13–36
- CACHE_ADD, cache action code standards, 13–37
- CACHE_ADD_UPDATE, cache action code standards, 13–38
- CACHE_CLOSE_CURSOR, cache action code standards, 13–39
- CACHE_DELETE, cache action code standards, 13–38
- CACHE_GET, cache action code standards, 13–37
- CACHE_GET_NEXT, cache action code standards, 13–39
- CACHE_TERMINATE, cache action code standards, 13–39
- CACHE_TERMINATE_ALL, cache action code standards, 13–39
- CACHE_UPDATE, cache action code standards, 13–38
- Calling
 - functions, ensure, 23–26
 - JDECACHE APIs, 13–5
 - work center initialization API, object librarian, 22–18
- Cancel, menu/toolbar items
 - Fix/Inspect form, B–18
 - Header Detail form, B–25
 - Headerless Detail form, B–32
- Cancel document (optional)
 - common parameters, 12–75
 - function name, 12–75
 - naming, transaction master business function modules, 12–75
 - special logic/processing required, 12–75
 - what does it do, 12–75
- Cell is exited after a change, B–49
- Changing
 - business function, parent DLL, 12–39
 - named mappings, C–11
 - processing option template, language, 14–14
 - row and column text for all applications, 3–21
 - size, forms, grids, and controls, 10–26
 - tab sequence, 10–92
- Check boxes, creating, 10–46
- Check in, confirming object, 8–3
- Checkout log
 - Object Librarian, 8–1
 - purging, 8–5
 - working with, 8–3
- Child nodes, 10–84
- Choosing
 - data items, business view, 5–9
 - data items for the table, 4–10
 - table, business view, 5–8
- Clear cache
 - function name, 12–74
 - naming, transaction master business function modules, 12–74
 - special logic/processing required, 12–74
 - what does it do, 12–74
- Clearing, cache, 13–36
- Clearing dialog
 - Fix/Inspect form, B–16
 - Header Detail form, B–21
 - Headerless Detail form, B–29
 - Search/Select form, B–37
- Close, menu/toolbar items
 - Find/Browse form, B–5
 - Parent/Child form, B–12
 - Search/Select form, B–37
- Closing, cursor, 13–19
- Closing form
 - Fix/Inspect form, B–17, B–37
 - Form/Browse form, B–5
 - Header Detail form, B–22
 - Headerless Detail form, B–29
 - Message form, B–39
 - Parent/Child form, B–12
- Coexistence
 - data item used in RPG program, 3–13
 - indices and logicals, 4–5
- Column, exited and changed - asynch, 11–117
- Column and row text for all applications, changing, 3–21
- Columnar and group sections, processing, 18–9
- Comment lines, event rules, adding, 11–39
- Commit (manual commit mode), 15–2
- Commits, 15–1
 - commit (manual commit mode), 15–2
 - rollback, 15–2
 - Two Phase, 15–2
- Common parameters
 - begin document, 12–68, 12–71
 - cancel document (optional), 12–75
 - edit document, 12–73

- end document, 12–74
- Compile section, build output, 12–52
- Components, business functions, 12–3
- Conditionalizing, level-break, headers or footers, 18–21
- Confirming, object has been checked in, 8–3
- Connecting, database, 12–28
- Considerations, processing options, language, 14–14
- Control
 - associating an item or description, 10–92
 - exited and changed - asynch, 11–117
 - exited processing, 11–12
- Control and static text, moving, 10–26
- Control is entered, Edit control, B–42
- Control is exited, Edit control, B–42
- Control is exited and changed asynch, Edit control, B–43
- Control is exited and changed inline, Edit control, B–43
- Control Modes, 10–107
- Control table modification rules, A–7
- Controls
 - aligning, 10–29
 - group, 10–29
 - alignment grid to align, 10–30
 - bitmap, creating, 10–66
 - changing size, 10–26
 - creating, UDC edit, 10–58
 - cut, copy, and paste, 10–29
 - disconnecting, static text, 10–60
 - media object, creating, 10–60, 10–63
 - moving, 10–25
 - paste, cut, and copy, 10–29
 - selecting, 10–25
 - size and place using the size command, 10–27
 - size using a mouse, 10–27
 - tree, 10–68
 - understanding, 11–2
 - working with, 10–43
- Conversion trigger, currency, creating, 16–7
- Copy, menu/toolbar items
 - Find/Browse form, B–6
 - Header Detail form, B–26
 - Headerless Detail form, B–33
 - Parent/Child form, B–13
 - Search/Select form, B–38
- Copying
 - event rules, 11–36
 - objects, 2–22
 - tables, 4–18
- Creating
 - assignment, 11–89
 - bitmap controls, 10–66
 - business function documentation, 12–87, 12–88
 - business function event rules, 12–7
 - check boxes, 10–46
 - currency, conversion trigger, 16–7
 - data structure, Data Dictionary item, 22–12
 - data structure documentation, 12–91
 - data structures, 6–9
 - business function, 6–12
 - media objects, 6–9
 - processing options, 6–11
 - edit controls, 10–54
 - event rule variables, 11–54
 - expression for an assignment, 11–91
 - form interconnections, 11–75
 - forms, 10–17
 - group boxes, 10–70
 - if and while statements, 11–46
 - level-break, business function, 22–15
 - level-break message, business function data structure, 22–13
 - master business function, processing options, 12–66
 - media object controls, 10–60, 10–63
 - Menu/Toolbar Exits, 10–33
 - modal form interconnect, 11–76
 - modeless form interconnect, 11–80
 - multi-level error messaging, 21–11
 - named mapping, C–7
 - parameter documentation, 12–94
 - processing options, data structure, 14–5
 - push buttons, 10–45
 - radio buttons, 10–48
 - report interconnections, 11–85
 - simple error message, 21–14
 - smart fields, C–7
 - static text controls, 10–52
 - subcategories in menus, 10–40
 - table event rules, 11–107
 - Table I/O event rule, 11–98
 - table join, 5–14
 - table union, 5–16
 - text substitution, error message, 21–16
 - UDC edit controls, 10–58
- Creating and saving, event rules, 11–31

- Creating data dictionary items, for level break messages, 22–9
- Creating level-break messages, 22–9
- Creating or editing, event rules for a business function event rules, 12–11
- Creating web applications, 24–4
- Cross reference facility, 7–1
 - working with, 7–3
- Cross reference information, rebuilding, 7–15
- Currency
 - advantages, 16–1
 - conversion, setting up, 16–5
 - conversion trigger, creating, 16–7
 - international, requirements, 16–1
 - OneWorld, implementation, 16–1
 - sensitive controls, showing, 16–6
 - working with, 16–3
- Currency processing, event rules, working with, 16–4
- Cursor
 - advancing APIs, 13–17
 - closing, 13–19
 - deleting records, 13–19
 - JDECACHE, 13–15
 - opening, 13–15
 - JDECACHE dataset, 13–16
 - multiple, JDECACHE, 13–20
 - resetting, 13–19
 - updating records, 13–18
- Cursor handle, data type, HJDECURSOR, 13–15
- Custom section
 - attaching, logic, 18–16
 - understanding, 18–15
 - using, 18–17
- Customizing, environment, Visual C++ debugging, 23–23
- Cut, copy, and paste, controls, 10–29
- Cutting, event rules, 11–36

D

- data, verifying, using the Universal Table Browser, 4–21
- Data Dictionary
 - message types, error, 21–2
 - messages, working with, 21–13
 - naming data items, 3–3

- OneWorld tools, 1–8, 3–1
 - runtime, how used, 3–2
 - storing, 3–3
 - data items, 3–3
 - types
 - error messages, 3–4
 - glossary items, 3–4
 - updating items for languages, 3–20
 - using, 3–7
- Data Dictionary item, data structure, creating, 22–12
- Data dictionary items
 - creating, for level-break messages, 22–9
 - naming conventions, 3–13
- Data Dictionary menu - GH951, 3–7
- Data Dictionary performance, performance tips, 25–4
 - overrides, 25–4
 - triggers, 25–4
 - validation, 25–5
- Data Dictionary triggers, override, design time, 10–97
- Data interdependence, 15–3
- Data Item, alias, 3–13
- Data item
 - interactive message
 - adding, 21–15, 21–17
 - attaching, 21–21
 - name, 3–12
- Data Items, searching, 7–4
- Data items
 - cache business functions, data structure, 13–37
 - choosing, business view, 5–9
- Data Dictionary
 - naming, 3–3
 - storing, 3–3
 - defining, 3–11
 - naming, 3–12
 - table, choosing, 4–10
- Data Items form, 3–20
- Data Retrieval, Fix/Inspect form, B–16
- Data retrieval
 - Find/Browse form, B–4
 - first level node in the tree, Parent/Child form, B–10
 - Header Detail form, B–21
 - Headerless Detail form, B–28
 - Parent/Child forms, B–10
 - Search/Select form, B–36

- tree node expand, Parent/Child form, B-11
 - Data selection and sequencing
 - performance tips, 25-10
 - processing time, 18-13
 - Data structure
 - business function, creating a level-break message, 22-13
 - creating
 - Data Dictionary item, 22-12
 - documentation, 12-91
 - data items, cache business functions, 13-37
 - documentation, template, 12-94
 - header file
 - Group cache business function, 13-39
 - individual cache business function, 13-41
 - preference profile, 13-39
 - Math_Numeric currency, application, 16-5
 - processing options, creating, 14-5
 - Data structure modification rules, A-8
 - Data structure objects, data structure
 - performance, performance tips, 25-10
 - Data structure performance, performance tips, 25-10
 - data structure objects, 25-10
 - restrict the number of fields, 25-10
 - Data structure template, message, adding, 21-20
 - Data Structures, searching, 7-9
 - Data structures
 - creating, 6-9
 - business function, 6-12
 - media objects, 6-9
 - processing options, 6-11
 - interconnect, working with, 6-3
 - OneWorld tools, 6-1
 - system generated, 6-1
 - form, 6-1
 - report, 6-1
 - user generated, 6-1
 - business function, 6-2
 - media objects, 6-1
 - processing options, 6-1
 - Data type
 - HJDECURSOR, cursor handle, 13-15
 - JDEDATE, 12-24
 - MATH_NUMERIC, 12-24
 - OneWorld, specific, 12-23
 - Database, connecting, 12-28
 - Database APIs, using, 12-25
 - Database communications steps, understanding, 12-28
 - Database items, 10-50
 - associating, 10-91
 - filtering, 10-57
 - inserting, form, 10-50
 - Database locking, 15-21
 - record is changed, 15-21
 - record locking, 15-22
 - Database Output, overriding environments, 19-7
 - Database output
 - defining, 19-3
 - Report Design Aid, table, 19-1
 - Date functions, advanced functions, expression manager, 11-93
 - DB2 for OS/400, index limitations for various databases, performance tips, 25-6
 - Debugging, 23-1
 - applications, 23-11
 - breakpoints, understanding, 23-9
 - business functions, 12-38
 - attached to batch applications, 23-18
 - attached to interactive applications, 23-15
 - Visual C ++, 23-15
 - Event Rules, working with, 23-5
 - information, enabling, 23-5
 - inspecting or modifying a variable, 23-12
 - log, 23-27
 - modifying or inspecting a variable, 23-12
 - process, overview, 23-1
 - strategies, 23-25
 - tracing, 23-29
 - turn on, 23-29
- Decimal trigger, overriding, 10-106
- Default flags
 - Find/Browse form, B-3
 - Fix/Inspect form, B-15
 - Header Detail form, B-19
 - Headerless Detail form, B-27
 - Parent/Child form, B-9
 - Search/Select form, B-35
- Default triggers
 - attaching, 3-23
 - understanding, 3-4
- Default value trigger, overriding, 10-99
- Default value triggers, attaching, 3-25
- Defining
 - active message, 21-29
 - cache data structure, 13-37

- data items, 3–11
- database output, 19–3
- form properties, 10–18
- general information, 3–16
- glossary text, runtime message, 21–19
- grid column properties, 10–74
- grid properties, 10–71
- indices, 4–11
- literal values in event rule logic, 11–50
- options
 - edit or UDC edit control, 10–90
 - form, 10–87
 - forms, grids, edit controls, 10–87
 - grid, 10–88
- properties, parent/child control, 10–77
- subsystem jobs, 20–3
- Transaction processing
 - form, 15–8
 - report, 15–19
- Defining OneWorld subsystem jobs, 20–3
 - Report Data Structure form, 20–5
 - Report Properties form, 20–4
- Delete
 - menu, 17–12
 - menu/toolbar items
 - Find/Browse form, B–6
 - Header Detail form, B–25
 - Headerless Detail form, B–32
 - Parent/Child form, B–12
 - Search/Select form, B–38
- Delete button, processing, 11–28
- Delete grid rec from DB after, runtime processing, 11–30
- Delete grid rec from DB before, runtime processing, 11–29
- Delete grid rec verity after, runtime processing, 11–29
- Delete grid rec verity before, runtime processing, 11–28
- Delete Object Specifications form, 2–22
- Deleting
 - objects, 2–21
 - records, cursor, 13–19
- Description
 - process flow for Edit control, B–41
 - process flow for Find/Browse form, B–3
 - process flow for Fix/Inspect form, B–15
 - process flow for Grid control, B–45
 - process flow for Header Detail form, B–19
 - process flow for Headerless Detail form, B–27
 - process flow for Message form, B–39
 - process flow for Parent/Child form, B–9
 - process flow for Search/Select form, B–35
- Description standards, business function, cache business function source description, 13–35
- Descriptions, associating, 10–91
- Design standards, J.D. Edwards, 10–5, 10–26, 11–72, 11–80
- Design time
 - how OneWorld works during, 1–13
 - override, data dictionary triggers, 10–97
- Designing
 - business function, 2–29
 - form layout, 10–25
- Detail, cache structure, 12–77
- Detail data selection and sequencing
 - Find/Browse form, B–4
 - Header Detail form, B–20
 - Headerless Detail form, B–28
 - Parent/Child form, B–10
 - Search/Select form, B–36
- Developing web applications, 24–1
- Development Cycle, understanding, 1–6
- Dialog, initialized, 11–16
- Dialog initialization
 - Find/Browse form, B–3
 - Fix/Inspect form, B–15
 - Header Detail form, B–19
 - Headerless Detail form, B–27
 - Message form, B–39
 - Parent/Child form, B–9
 - Search/Select form, B–35
- Dictionary items, 10–50
 - associating, 10–91
 - inserting, form, 10–51
- Disconnecting, static text from controls, 10–60
- Display rule triggers, attaching, 3–31
- Displaying
 - reference list of business functions, 2–28
 - type definitions, 6–7
- Distributed business functions, 12–5
- DLLs, 12–4
- Do in while loop, runtime processing, 11–18
- Do_Section event, batch events, 18–23
- Document processing, transaction master business function, implementing, 12–80

Documentation

- business function, 12–87
 - creating, 12–87, 12–88
 - template, 12–90
 - business function - values to pass, viewing, 12–99
 - business function document viewer, viewing, 12–100
 - business function search, viewing, 12–98
 - data structure
 - creating, 12–91
 - template, 12–94
 - generating, business function, 12–95
 - parameter, creating, 12–94
 - viewing, business function, 12–98
- Dragging and dropping, 10–86
- Dynamic Overrides, 11–113

E

Edit Control

- cell is exited after a change, B–49
- control exited and change asynch, B–43
- control exited and change inline, B–43
- control is entered, B–42
- control is exited, B–42
- double click on a grid row, B–49
- key pressed, B–49
- kill focus on grid, B–48
- process flow, description, B–41, B–45
- properties/options, B–41, B–46
- row is entered, B–48
- row is exited, B–48
- row is exited and changed (asynchronous portion), B–48
- row is exited validation, B–48
- set focus on grid, B–47

Edit controls

- creating, 10–54
- defining options, 10–90

Edit document

- application specific parameters, 12–73
- common parameters, 12–73
- function name, 12–72
- hook up tips, 12–72
- naming, transaction master business function modules, 12–72
- special logic/processing required, 12–72

- what does it do, 12–72

Edit line

- naming, transaction master business function modules, 12–70
- what does it do, 12–70

Edit rule trigger, overriding, 10–102

Edit rule triggers, attaching, 3–28

Editing keypad, expression manager, 11–93

Editing or creating

- event rules, business function event rules, 12–11
- event rules for a business function event rules, 12–11

Elements, forms, 10–3

Embedded

- Event Rules, OneWorld tools, 1–11
- event rules, 11–3

Enabling, debugging information, 23–5

Encountering, errors, application, 23–26

End document

- application specific parameters, 12–74
- common parameters, 12–74
- function name, 12–73
- hook up tips, 12–74
- naming, transaction master business function modules, 12–73
- special logic/processing required, 12–74
- what does it do, 12–73

End_Level–Break Footer, section events, 18–22

End_Level–Break Header, section events, 18–22

Ending, unexpectedly, program, 23–25

Ensure, functions being called, 23–26

Environment, Visual C++ debugging, customizing, 23–23

Environments, Report Design Aid, override, 19–7

Error

- application, encountering, 23–26
- automatic setting, 21–7
- JDECACHE, 13–23
 - references, 13–23
- manual setting, 21–7
 - business function API, 21–8
 - system function, 21–7
- message types, 21–2
- resetting, 21–9
 - Find button, 21–10
 - OK button, 21–9

- setting, 21–7
- Error code
 - application, field to highlight, 21–6
 - error set, 21–6
- Error handling, understanding, 21–5
- Error message
 - batch, 22–1
 - creating simple, 21–14
 - text substitution, creating, 21–16
 - Work Center API, 22–7
- Error messages
 - Data Dictionary, types, 3–4
 - text substitution, 22–5
- Error messaging performance, performance tips, 25–15
- Error output, reviewing, 12–61
- Error set, error code, 21–6
- Error setting, event ID, how used, 21–6
- Errors, resolving, 12–62
 - business function builder tools, 12–62
- Event
 - attaching a business function, 11–69
 - attaching a system function, 11–61
- Event Flow, group section, 18–34
- Event flow, typical, Find/Browse form, 11–14
- Event ID, error setting, how used, 21–6
- Event rule
 - form flow, 11–14
 - manipulation, 11–12
 - Table I/O, creating, 11–98
- Event rule business functions, naming conventions, 2–15
- Event Rule Design, working with, 11–31
- Event rule logic
 - if statements, 11–45
 - literal values, defining, 11–50
 - printing, 11–40
 - while statements, 11–46
- Event rule modification rules, A–8
- Event rule variables
 - batch application, 11–54
 - creating, 11–54
 - interactive, 11–53
 - working with, 11–53
- Event Rules
 - debugging, working with, 23–5
 - OneWorld tools, 1–10
 - business function, 1–11
 - embedded, 1–11
- Event rules
 - adding, comment lines, 11–39
 - application, 11–4
 - attaching, Menu/Toolbar Exit, 10–38
 - business function, 11–3
 - business function event rules, creating or editing, 12–11
 - copying, 11–36
 - creating and saving, 11–31
 - currency processing, working with, 16–4
 - cutting, 11–36
 - embedded, 11–3
 - paste, 11–36
 - searching, 7–12
 - table, 11–4
 - understanding, 11–3
 - validating, 11–42
- Event rules buttons, understanding, 11–4
- Event Rules Debugger
 - application tree control, 23–8
 - breakpoint manager, 23–9
 - event rules window, 23–8
 - search combo box, 23–10
 - variable selection and display control, 23–9
- Event Rules Design
 - OneWorld tools, 11–1
 - Table I/O, 11–97
- Event rules performance, performance tips, 25–12
 - I/O commands in event rule code, 25–13
 - JDB API calls in a business function, 25–13
 - JDBFetch statements, 25–13
 - JDBFetchNext, 25–13
 - jdeCache instead of work tables, 25–13
- Event Rules Window, Event Rules Debugger, 23–8
- Event-driver, model, interactive message, 21–5
- Event-level, interactive event rule variables, 11–54
- Events
 - batch
 - system functions within, 18–31
 - understanding, 18–23
 - level-break section, 18–21
 - understanding, 11–2
- Example program, JDECACHE, 13–25
- Existing error message, locating, 21–13
- Existing message, locating, 21–13
- Exited and changed - asynch
 - Column, 11–117

- control, 11–117
- row, 11–117
- Exited processing, control, 11–12
- Expanded node, BrowsER, 11–125
- Expression, creating for an assignment, 11–91
- Expression editing field, expression manager, 11–93
- Expression manager, 11–92
 - advanced functions, 11–93
 - date functions, 11–93
 - general functions, 11–94
 - text functions, 11–94
 - trig functions, 11–94
- assignment display, 11–92
- available information, 11–93
- Editing keypad, 11–93
- expression editing field, 11–93
- Extending, transaction boundary, 15–9
 - between forms, 15–9
 - business functions, 15–13
 - Table I/O, 15–16
- External Business Function, calling an API from, 12–29
- External data dictionary items, naming conventions, 3–14

F

- F0002 - Next Number table, 3–36
- F9860 - Object Librarian Master Table, 2–2
- F9861 - Object Librarian Detail Table, 2–3
- F9862 - Object Librarian Function Detail, 2–3
- F9863 - Object Librarian-Business Function Object Relationships, 2–4
- F9865 - Form Information Table, 2–5
- Fast path, create selection, 17–22, 17–23
- Features, business functions, 12–2
- Field relationships, viewing, 7–14
- Filter, menus, 17–1
- Filter ER Records, BrowsER, 11–126
- Filtering, database items, 10–57
- Find, menu/toolbar items
 - Find/Browse form, B–5
 - Header Detail form, B–25
 - Headerless Detail form, B–32
 - Parent/Child form, B–13
 - Search/Select form, B–37

- Find button, error, resetting, 21–10
- Find/Browse
 - forms, 10–6
 - OneWorld tools, Form Design, 1–9
- Find/browse, form design, performance tips, 25–11
- Find/Browse form
 - closing form, B–5
 - data retrieval, B–4
 - default flags, B–3
 - dialog initialization, B–3
 - header data retrieval form, B–4
 - menu/toolbar items, B–5
 - add, B–6
 - close, B–5
 - copy, B–6
 - delete, B–6
 - find, B–5
 - select, B–5
 - user defined items, B–6
 - process flow, B–3
 - description, B–3
 - typical event flow, 11–14
- Find/browse form, detail data selection, B–4
- First level node in the tree, data retrieval, Parent/Child form, B–10
- Fix/Inspect
 - forms, 10–7
 - OneWorld tools, Form Design, 1–9
- Fix/Inspect form
 - clearing dialog, B–16
 - closing form, B–17
 - data retrieval, B–16
 - default flags, B–15
 - dialog initialization, B–15
 - Find/Browse form, OK, B–17
 - menu/toolbar items, B–17
 - cancel, B–18
 - OK, B–17
 - user defined items, B–18
 - process flow, B–15
 - description, B–15
- Footer sections, level-break, 18–21
- Form
 - Add Object, 2–8
 - Data Items, 3–20
 - Delete Object Specifications, 2–22
 - Object Librarian Codes, 2–24
 - Work With Object Librarian, 2–18

- Form controls, understanding, 10–44
- Form data structures
 - modifying, 6–3
 - system generated, 6–1
- Form Design, OneWorld tools, 1–9
 - Find/Browse, 1–9
 - Fix/Inspect, 1–9
 - Header Detail, 1–9
 - Headerless Detail, 1–9
 - Message, 1–10
 - Parent/Child, 1–9
 - Search and Select, 1–9
- Form design, performance tips, 25–11
 - find/browse, 25–11
 - header detail, 25–11
 - headerless detail, 25–11
- Form flow, event rule, 11–14
- Form Information Table - F9865, 2–5
- Form Interconnect calls, J.D. Edwards, standards, B–39, B–43, B–50
- Form interconnections
 - creating, 11–75
 - naming conventions, 2–15
- Form interconnects
 - modal processing, 11–75
 - modeless processing, 11–75
- Form processing, B–1
 - understanding, 11–2
- Form types, understanding, 10–5
- Format tables, performance impact, 12–84
- Formatted data, universal table browser, example, 4–25, 4–26
- Formatting trigger, overriding, 10–101
- Form-level, interactive event rule variables, 11–53
- Forms
 - Add Object, 12–42
 - business function builder, understanding, 12–45
 - Business Function Design, 12–44
 - changing size, 10–26
 - controls, understanding, 11–2
 - creating, 10–17
 - defined, 10–2
 - defining options, 10–87
 - defining properties, 10–18
 - designing layout, 10–25
 - elements, 10–3
 - events, understanding, 11–2
 - Find/Browse, 10–6
 - Fix/Inspect, 10–7
 - form processing, understanding, 11–2
 - generating, 24–9
 - Header Detail, 10–9
 - Headerless Detail, 10–10
 - inserting
 - database items, 10–50
 - dictionary items, 10–51
 - Java Generator B733, 24–9
 - Menu Execution, 17–22
 - Menu Header Revisions, 17–4, 17–16, 17–27
 - Menu Selection Revisions, 17–10, 17–12, 17–13, 17–17, 17–18, 17–20, 17–21, 17–25, 17–28
 - Menu Selection Search, 17–26
 - Menu Text Overrides, 17–27, 17–28
 - Message, 10–13
 - naming conventions, 2–14, 10–20
 - OneWorld Report, 17–16
 - Parent/Child, 10–14
 - parent/child control, 10–76
 - Renumber Selection, 17–29
 - Report Data Structure form, 20–5
 - Report Properties, 20–4
 - Search and Select, 10–12
 - searching, 7–11
 - select type, 10–17
 - testing, 10–127
 - types, understanding, 10–5
 - User Defined Codes, 17–24
 - Windows Application, 17–19
 - Work With Menu Selections, 17–10, 17–23, 17–25, 17–28, 17–29
 - Work With Menus, 17–26
 - Work With Object Librarian, 12–42
 - Work With User Defined Codes, 17–23
 - WorldVision, 17–17
- Forms Design, OneWorld tools, 10–1
- Function
 - calling, ensure, 23–26
 - send message system, use, 21–26
- Function name
 - begin document, 12–67, 12–70
 - cancel document (optional), 12–75
 - clear cache, 12–74
 - edit document, 12–72
 - end document, 12–73
 - source file, cache business function description, 13–35

Functions

- attaching, 11–61
- distributed business, 12–5

Functions list, business function builder form, 12–46

G

General Functions, advanced functions, expression manager, 11–94

General information, defining, 3–16

General standards, programming standards, cache, 13–36

Generating

- business view, 5–16
- OneWorld applications, 24–7
- tables, 4–13
- web applications, 24–7

Generating a form, 24–9

Generating all applications, 24–15

Generating an application, 24–14

Generator

- deleting data sources, 24–8
- generating by user, 24–8
- messages, 24–8
- multiple language, 24–8
- overview, 24–7
- status window, 24–8
- stopping the generation, 24–8
- using, 24–3

GH951 - Data Dictionary menu, 3–7

Glossary

- adding for languages, 3–41
- updating, 3–40

Glossary items, Data Dictionary, types, 3–4

Glossary text, runtime message, defining, 21–19

Go command, Visual C++ debugger, useful features, 23–21

Grid

- defining options, 10–88
- processing, 11–13
- set lower limits, 11–13

Grid column properties, defining, 10–74

Grid properties, defining, 10–71

Grid rec is fetched, runtime processing, 11–19

Grid row

- double click, B–49

key pressed, B–49

Grids, changing size, 10–26

Group, controls, aligning, 10–29

Group and columnar sections, processing, 18–9

Group boxes, creating, 10–70

Group cache business function, data structure, header file, 13–39

Group Section, event flow, 18–34

H

Handle request, event rule variable, 11–58

HCACHE, 13–4

Header, cache structure, 12–76

Header data retrieval

- Find/Browse forms, B–4
- Header Detail form, B–20
- Parent/Child form, B–10
- Search/Select form, B–36

Header Detail

- forms, 10–9
- OneWorld tools, Form Design, 1–9

Header detail, form design, performance tips, 25–11

Header Detail form

- Add entry row to grid, B–22
- clearing dialog, B–21
- closing form, B–22
- data retrieval, B–21
- default flags, B–19
- detail data selection and sequencing, B–20
- dialog initialization, B–19
- header data retrieval form, B–20
- menu/toolbar items, B–22

add, B–26

cancel, B–25

copy, B–26

delete, B–25

find, B–25

OK, B–22

user defined items, B–26

process flow, B–19

description, B–19

Header file, data structure

- Group cache business function, 13–39
- individual cache business function, 13–41
- preference profile, 13–39

Header file sections, 12–16

Header sections, level-break, 18–21
 page breaks, 18–21
 Headerless Detail
 forms, 10–10
 OneWorld tools, Form Design, 1–9
 Headerless detail, form design, performance tips, 25–11
 Headerless Detail form
 add entry row to grid, B–29
 clearing dialog, B–29
 closing form, B–29
 data retrieval, B–28
 default flags, B–27
 detail data selection and sequencing, B–28
 dialog initialization, B–27
 menu/toolbar items, B–30
 add, B–33
 cancel, B–32
 copy, B–33
 delete, B–32
 find, B–32
 OK, B–30
 user defined items, B–33
 process flow, B–27
 description, B–27
 Hide object, batch events, 18–32
 Highlights, source code, sample, 22–15
 HJDECURSOR, 13–4
 data type, cursor handle, 13–15
 Hook up tips
 begin document, 12–68
 edit document, 12–72
 end document, 12–74
 How OneWorld works at design time, 1–13
 Hungarian notation, example of, 11–58

I

I/O commands in event rule code, event rules performance, performance tips, 25–13
 If and while statements
 creating, 11–46
 working with, 11–45
 If statements, event rule logic, 11–45
 imaging, third-party software, 10–117
 Implementation, OneWorld, currency, 16–1
 Implementing, transaction master business functions, 12–79
 document processing, 12–80
 single record processing, 12–79
 Incorrect, output of program, 23–26
 Index
 defined, 4–1
 naming, J.D. Edwards naming standards, 4–13
 using to access the cache, 13–9
 Index considerations
 limits on row set size, performance tips, 25–5
 table design performance, performance tips, 25–5
 Index limitations for various databases, performance tips, 25–6
 Access32, 25–6
 DB2 for OS/400, 25–6
 key column violation, 25–7
 Oracle, 25–7
 specification file corruption, 25–8
 SQLSERVER, 25–6
 Indices, 5–2
 defining, 4–11
 JDECACHE, setting up, 13–6
 naming conventions, 4–5
 Indices and logicals, coexistence, 4–5
 Individual cache business function, data structure, header file, 13–41
 Information
 debugging, enabling, 23–5
 message types, 21–3
 Information structure, standard parameters, 12–83
 INIT_Level Break Footer Section event, level-break footer, 18–22
 INIT_Level Break Header Section event, level-break header, 18–22
 Initialized
 dialog, 11–16
 post dialog, 11–17
 pre dialog, 11–15
 Initializing, cache, 13–7
 Inserting
 database items, form, 10–50
 dictionary items, form, 10–51
 Inspecting or modifying, variable, debugging, 23–12
 Interactive application, adding, 9–5

- Interactive application modification rules, A-4
- Interactive Applications, searching, 7-5
- Interactive applications, business functions, debugging, 23-15
- Interactive event rule variables, 11-53
 - event-level, 11-54
 - form-level, 11-53
- Interactive message
 - data item
 - adding, 21-15, 21-17
 - attaching, 21-21
 - event-driver model, 21-5
- Interconnect data structures, working with, 6-3
- International, currency, requirements, 16-1

J

- J.D. Edwards
 - design standards, 4-13, 10-5, 10-26, 11-72, 11-80
 - naming standards
 - application, 9-6, 10-20
 - business view, 5-4
 - processing option, 14-10
 - tables, 4-4
 - ODBC - Open database connectivity, 12-26
 - open database connectivity, 12-26
 - standards
 - form interconnect calls, B-50
 - from interconnect calls, B-39, B-43
- Java Generator B733 form, 24-9
- JDB API calls in a business function, event rules performance, performance tips, 25-13
- JDBFetch statements, event rules performance, performance tips, 25-13
- JDBFetchNext, event rules performance, performance tips, 25-13
- JDEBASE API, standard categories, 12-27
- JDECACHE
 - API set, 13-3
 - cursor, 13-15
 - opening, 13-15
 - cursor advancing APIs, 13-17
 - error, 13-23
 - references, 13-23
 - example program, 13-25
 - JDEKRNL, 13-2

- multiple cursor support, 13-20
- non-cursor advancing APIs, 13-18
- partial key, 13-21
- performance considerations, 13-2
- setting up indices, 13-6
- when to use, 13-1
- working with, 13-5
- JDECACHE APIs, calling, 13-5
- JDECACHE cursor APIs, types, 13-17
- JDECACHE dataset, cursor, 13-16
- jdeCache instead of work tables, event rules performance, performance tips, 25-13
- JDECACHE management APIs, manage, 13-3
- JDECACHE Manipulation APIs, 13-4
 - HCACHE, 13-4
 - HJDECURSOR, 13-4
- JDECACHE partial key
 - how works, 13-21
 - JDECACHE, 13-21
- jdeCacheFetchPosition API, searching, 13-19
- jdeCacheInit/jdeCacheTerminate rule, using, 13-12
- JDEDATE, data type, 12-24
- JDEKRNL, JDECACHE, 13-2
- Jobs, subsystem, defining, 20-3
- Joined business view versus table I/O, business view performance, performance tips, 25-10
- Joined sections, attaching logic, 18-12
- Joined views, naming conventions, 2-14, 5-5

K

- Key column violation, index limitations for various databases, performance tips, 25-7
- Kill focus on grid, B-48

L

- Language
 - media objects, considerations, 10-121
 - processing options, change template, 14-14
- Language considerations
 - media objects, 10-121
 - processing options, 14-14
- Language Preview, 10-128

- Language specific media object attachment, adding, 10–122
 - Last grid rec has been read, runtime processing, 11–24
 - Level, message types, 21–2
 - Level-break
 - conditionalizing, header or footer, 18–21
 - creating, business function, 22–15
 - footer sections, 18–21
 - header sections, 18–21
 - page breaks, 18–21
 - message, 22–3
 - Level-break footer, INIT_Level Break Footer Section event, 18–22
 - Level-break footers or headers, conditionalizing, 18–21
 - Level-break header, INIT_Level Break Header Section event, 18–22
 - Level-break headers or footers, conditionalizing, 18–21
 - Level-break message
 - business function data structure, creating, 22–13
 - understanding, 22–6
 - Level-break messages
 - creating, 22–9
 - creating a data dictionary item for, 22–9
 - Level-break processing, understanding, 18–19
 - Level-break section, events, 18–21
 - Limits on row set size, index considerations, performance tips, 25–5
 - Link section, build output, 12–53
 - Linking, business function objects, 12–49
 - Literal values, event rule logic, defining, 11–50
 - Loading, nodes to the tree, 10–83
 - Locals window, Visual C++ debugger, useful features, 23–22
 - Locating
 - existing error message, 21–13
 - existing message, 21–13
 - Log
 - debugging, 23–27
 - SQL, tracing, 23–28
 - Logic
 - custom section, attaching, 18–16
 - event rule, printing, 11–40
 - Logs, checkout
 - purging, 8–5
 - working with, 8–3
- ## M
- Makefile section, build output, 12–52
 - Manage cache, JDECACHE, 13–3
 - Manipulation, event rule, 11–12
 - Manual error, error setting, 21–7
 - business function API, 21–8
 - system function, 21–7
 - Master business functions
 - creating, processing options, 12–66
 - information structure, standard parameters, 12–83
 - master file, 12–81
 - naming convention, 12–65
 - Master file, master business functions, 12–81
 - MATH_Numeric, data type, 12–24
 - Math_Numeric currency, data structure, application, 16–5
 - Media object controls, creating, 10–60, 10–63
 - Media object data structures
 - creating, 6–9
 - user generated, 6–1
 - Media objects
 - media considerations, 10–121
 - processing, 10–117
 - standard processing, using, 10–118
 - Memory, sections, OneWorld allocates, 18–13
 - Memory allocation, business function performance, performance tips, 25–14
 - Menu
 - Data Dictionary menu - GH951, 3–7
 - delete, 17–12
 - review selections, 17–6
 - Menu Design tables, 17–2
 - F0082 (Menu Master), 17–2
 - F00821 (Menu Selection), 17–2
 - F0083 (Menu Text Override), 17–2
 - F0084 (Menu Path), 17–2
 - Menu selection, 17–9
 - add or change a menu selection, 17–9
 - changing text, 17–28
 - copy a menu selection, 17–25
 - define the menu selection, 17–12
 - name a menu selection, 17–10
 - Menu selection revisions, 17–25
 - Menu selections, renumbering a menu selection, 17–28

Menu text, change menu text for languages, 17–26

Menu/Toolbar Exits

- attaching, event rules, 10–38

- creating, 10–33

Menu/toolbar items

- Find/Browse form, B–5

 - add, B–6

 - close, B–5

 - copy, B–6

 - delete, B–6

 - find, B–5

 - select, B–5

 - user defined items, B–6

- Fix/Inspect form, B–17

 - cancel, B–18

 - user defined items, B–18

- Header Detail form, B–22

 - add, B–26

 - cancel, B–25

 - copy, B–26

 - delete, B–25

 - find, B–25

 - OK, B–22

 - user defined items, B–26

- Headerless Detail form, B–30

 - add, B–33

 - cancel, B–32

 - copy, B–33

 - delete, B–32

 - find, B–32

 - OK, B–30

 - user defined items, B–33

- Parent/Child form, B–12

 - add, B–13

 - close, B–12

 - copy, B–13

 - delete, B–12

 - find, B–13

 - select, B–12

 - user defined items, B–13

- Search/Select, add, B–38

- Search/Select form, B–37

 - copy, B–38

 - delete, B–38

 - find, B–37

 - select, B–37

 - user defined items, B–38

Menus

- add a OneWorld report section, 17–15

- add a Windows application, 17–18

- add a WorldVision application, 17–17

- add an application, 17–13

- create subheading, 17–21

- creating subcategories, 10–40

- defining new menus, 17–3

- design menus, 17–1

- filtering menus, 17–1

- link another menu, 17–21

- OneWorld tools, 1–12

- print report, 17–6

- understanding menus, 17–1

- working with menus, 17–3

Message

- action, 22–5

- active, defining, 21–29

- application, 21–1

- batch error, understanding, 22–3

- data structure template, attaching, 21–20

- error

 - batch, 22–1

 - creating a text substitution, 21–16

 - creating simple, 21–14

- forms, 10–13

- level-break, 22–3

 - understanding, 22–6

- locating

 - existing, 21–13

 - existing error, 21–13

- OneWorld tools, Form Design, 1–10

- runtime, defining the glossary text, 21–19

- types, 21–1

 - error, 21–2

 - information, 21–3

 - level, 21–2

 - workflow, 21–2

Message form

- Buttons, B–39

- closing form, B–39

- dialog initialization, B–39

- process flow, B–39

 - description, B–39

Messages

- error, text substitution, 22–5

- level-break, creating, 22–9

- working with, 21–13

Modal form interconnect, creating, 11–76

Modal processing, form interconnects, 11–75

Model, event-driven, interactive message, 21–5

Modeless form interconnect, creating, 11–80

Modeless processing, form interconnects, 11–75
 Modification rules
 Application text, A–6
 business functions, A–9
 business views, A–7
 control tables, A–7
 data structure, A–8
 event rules, A–8
 interactive applications, A–4
 OneWorld, A–1
 Reports, A–5
 table specifications, A–6
 understanding, report specifications, A–5
 versions, A–10
 Modifying
 form data structures, 6–3
 objects, 2–17
 report data structures, 6–5
 Modifying or inspecting, variable, debugging, 23–12
 Mouse, size controls, 10–27
 Moving
 control and static text, 10–26
 controls, 10–25
 Multi-error messaging, create, 21–11
 Multi-level error messaging, C business functions, 21–11
 Multiple cursor support, JDECACHE, 13–20
 Multiple-table or single-table, business view performance, performance tips, 25–9

N

Named ER, naming conventions, 2–15
 Named Mapping, creating, C–7
 Named mappings, changing, C–11
 Naming
 Data Dictionary, data items, 3–3
 data items, 3–12
 index, J.D. Edwards naming standards, 4–13
 objects, 2–12
 transaction master business function modules, 12–66
 begin document, 12–67
 Cancel document (optional), 12–75
 clear cache, 12–74
 Edit document, 12–72

 edit line, 12–70
 end document, 12–73
 Naming convention, 12–84
 master business functions, 12–65
 Naming conventions
 applications, 2–14
 business functions, 2–16
 business views, 2–13
 data dictionary items, 3–13
 data structures, 2–16
 event rule business functions, 2–15
 external data dictionary items, 3–14
 form interconnections, 2–15
 forms, 2–14, 10–20
 indices, 4–5
 joined views, 2–14, 5–5
 named ER, 2–15
 processing option data structure, 14–10
 processing options, 14–11
 tables, 2–12
 Next number trigger, overriding, 10–104
 Next number triggers, attaching, 3–36
 Next Numbers table - F0002, 3–36
 Node, expanded, BrowsER, 11–125
 Nodes
 child, 10–84
 loading to the tree, 10–83
 parent, 10–84
 Non-formatted data, universal table browser, example, 4–24
 Non-cursor, advancing APIs, 13–18

O

Object
 confirming check in, 8–3
 Object Librarian, adding, 2–8
 Object ID, show, BrowsER, 11–125
 Object Librarian
 adding an object, 2–8
 checkout log, 8–1
 OneWorld tools, 1–8, 2–1
 tracking objects, 2–2
 working with, 2–7
 Object librarian, calling, work center initialization API, 22–18
 Object Librarian Codes, working with, 2–23
 Object Librarian Codes form, 2–24

- Object Librarian Detail Table - F9861, 2–3
- Object Librarian Function Detail - F9862, 2–3
- Object Librarian Master Table - F9860, 2–2
- Object Librarian-Business Function Object Relationships - F9863, 2–4
- Object Types, 2–12
- Objects
 - copying, 2–22
 - deleting, 2–21
 - modifying, 2–17
 - naming, 2–12
 - searching, 7–3
 - selecting from available, 6–4
 - understanding, 1–3
 - understanding how OneWorld stores, 1–7
- ODBC - Open database connectivity, J.D. Edwards, 12–26
- OK, menu/toolbar items
 - Header Detail form, B–22
 - Headerless Detail form, B–30
- Ok, menu/toolbar items, Fix/Inspect form, B–17
- OK button, error, resetting, 21–9
- OneWorld
 - allocates, memory for sections, 18–13
 - batch processes, subsystem jobs, 20–1
 - cache, 13–1
 - currency implementation, 16–1
 - design time, 1–13
 - modification rules, A–1
 - specific data type, 12–23
 - Transaction Processing, 15–3
 - Understanding how objects are stored, 1–7
- OneWorld report section, add to a menu, 17–15
- OneWorld subsystem jobs, defining for OneWorld, 20–3
- OneWorld tools, 1–1
 - Business Function Design, 1–10
 - Business functions, 12–1
 - Business View Design, 1–9, 5–1
 - Data Dictionary, 1–8, 3–1
 - Data structures, 6–1
 - Event Rules, 1–10
 - business function, 1–11
 - embedded, 1–11
 - Event Rules Design, 11–1
 - Form Design, 1–9
 - Find/Browse, 1–9
 - Fix/Inspect, 1–9
 - Header Detail, 1–9
 - Headerless Detail, 1–9
 - Message, 1–10
 - Parent/Child, 1–9
 - Search and Select, 1–9
 - Forms Design, 10–1
 - Menus, 1–12
 - Object Librarian, 1–8, 2–1
 - tracking objects, 2–2
 - Processing Options, 1–11
 - Report Design, 1–10
 - Table Design, 1–9, 4–1
 - Understanding, 1–8
- Open database connectivity, J.D. Edwards, 12–26
- Opening, JDECACHE cursor, 13–15
- Optimistic Locking, 15–21
- Options
 - BrowsER, working with, 11–124
 - defining, 10–87
 - edit or UDC edit control, 10–90
 - form, 10–87
 - grid, 10–88
- Oracle, index limitations for various databases, performance tips, 25–7
- Output, program, incorrect, 23–26
- Override
 - data dictionary triggers, design time, 10–97
 - decimal trigger, 10–106
 - default value trigger, 10–99
 - edit rule trigger, 10–102
 - formatting trigger, 10–101
 - next number trigger, 10–104
 - previously defined, 10–97
 - Report Design Aid, environments, 19–7
 - styles trigger, 10–105
 - user defined codes trigger, 10–103
 - visuals assist trigger, 10–100
- Overrides
 - Data Dictionary performance, performance tips, 25–4
 - dynamic, 11–113
- Overview, debugging, process, 23–1

P

- Page breaks, level-break headers, using with, 18–21
- Page-at-a-time processing, runtime processing, 11–18
- Parameter, creating, documentation, 12–94
- Parent DLL, business functions, changing, 12–39
- Parent nodes, 10–84
- Parent/Child
 - forms, 10–14
 - OneWorld tools, Form Design, 1–9
- Parent/child control
 - defining properties, 10–77
 - form, 10–76
- Parent/Child form
 - closing form, B–12
 - default flags, B–9
 - detail data selection, B–10
 - dialog initialization, B–9
 - header data retrieval form, B–10
 - menu/toolbar items, B–12
 - add, B–13
 - close, B–12
 - copy, B–13
 - delete, B–12
 - find, B–13
 - select, B–12
 - user defined items, B–13
 - process flow, B–9
 - description, B–9
- Parent/Child forms, data retrieval, B–10
 - first level node in the tree, B–10
 - tree node expand, B–11
- Paste, event rules, 11–36
- Paste options, setting, 11–37
- Paste, cut, copy, controls, 10–29
- Performance, applications, 25–1
- Performance considerations, JDECACHE, 13–2
- Performance impact, format tables, 12–84
- Performance tips, 25–3
 - applications, things to look for, 25–3
 - batch application performance, 25–12
 - setting up level breaks, 25–12
 - business function performance, 25–14
 - balance table closes and opens, 25–15
 - balance table opens and closes, 25–15
 - memory allocation, 25–14
 - business view performance, 25–9
 - joined business view versus table I/O, 25–10
 - restrict the number of fields, 25–9
 - single-table or multiple-table, 25–9
 - table I/O versus joined business view, 25–10
 - unions, 25–9
- Data Dictionary performance, 25–4
 - overrides, 25–4
 - triggers, 25–4
 - validation, 25–5
- data selection and sequencing, 25–10
- data structure performance, 25–10
 - data structure objects, 25–10
 - restrict the number of fields, 25–10
- error messaging performance, 25–15
- event rules performance, 25–12
 - I/O commands in event rule code, 25–13
 - JDB API calls in a business function, 25–13
 - JDBFetch statements, 25–13
 - JDBFetchNext, 25–13
 - jdeCache instead of work tables, 25–13
- form design, 25–11
 - find/browse, 25–11
 - header detail, 25–11
 - headerless detail, 25–11
- index considerations, limits on row set size, 25–5
- index limitations for various databases, 25–6
 - Access32, 25–6
 - DB2 for OS/400, 25–6
 - key column violation, 25–7
 - Oracle, 25–7
 - specification file corruption, 25–8
 - SQLSERVER, 25–6
- sequencing and data selection, 25–10
- table design performance, 25–5
 - index considerations, 25–5
 - World coexistence considerations, 25–6
- table I/O objects, 25–8
- Performing, build all, 12–54
- Pessimistic Locking, 15–22
- Portability and standards, understanding, 12–25
- Post dialog, initialized, 11–17
- Pre dialog, initialized, 11–15
- Preference profile cache, data structure, header file, 13–39

- Preserve and replace with an upgrade, A-3
- Previewing, tables, 4-17
- Previously defined triggers, overriding, 10-97
- Print, menu report, 17-6
- Printing, event rule logic, 11-40
- Problems
 - possible source, 23-26
 - source, 23-26
- Process, debugging, overview, 23-1
- Process flow for Edit Control, description, B-41
- Process flow for Find/Browse form, B-3
 - description, B-3
- Process flow for Fix/Inspect form, B-15
 - description, B-15
- Process flow for Grid Control, description, B-45
- Process flow for Header Detail form, B-19
 - description, B-19
- Process flow for Headerless Detail form, B-27
 - description, B-27
- Process flow for Message form, B-39
 - description, B-39
- Process flow for Parent/Child form, B-9
 - description, B-9
- Process flow for Search/Select form, B-35
 - description, B-35
- Processing
 - add button, 11-27
 - available batch objects, 18-34
 - available objects, 11-12
 - batch runtime, 18-33
 - delete button, 11-28
 - media objects, 10-117
 - runtime, 11-9
 - sections
 - group and columnar, 18-9
 - tabular, 18-11
 - select button, 11-25
- Processing Option, adding, J.D. Edwards naming standards, 14-10
- Processing Options, OneWorld tools, 1-11
- Processing options
 - data structure, creating, 14-5
 - defined, 14-1
 - language considerations, 14-14
 - master business functions, creating, 12-66
 - template, 14-2
 - attaching, 14-17
 - change language, 14-14

- Processing options data structures
 - creating, 6-11
 - user generated, 6-1
- Processing Options Template, for text translation, 14-14
- Processing time
 - data selection and sequencing, 18-13
 - sequencing and data selection, 18-13
- Program
 - ending unexpectedly, 23-25
 - output, incorrect, 23-26
- Programs and IDs, P0082 (Menu Design)
 - Menu Execution, 17-22
 - Menu Header Revisions, 17-4, 17-16, 17-27
 - Menu Selection Revisions, 17-10, 17-12, 17-13, 17-17, 17-18, 17-20, 17-21, 17-25, 17-28
 - Menu Selection Search, 17-26
 - Menu Text Overrides, 17-27, 17-28
 - OneWorld Report, 17-16
 - ReNUMBER Selection, 17-29
 - User Defined Codes, 17-24
 - Windows Application, 17-19
 - Work With Menu Selections, 17-10, 17-23, 17-25, 17-28, 17-29
 - Work With Menus, 17-26
 - Work With User Defined Codes, 17-23
 - WorldVision, 17-17
- Promotion Manager, 2-33
- Properties
 - defining, parent/child control, 10-77
 - grid, defining, 10-71
 - grid column, defining, 10-74
- Properties/Options, B-46
 - edit control, B-41
- Purging, checkout log, 8-5
- Push buttons, creating, 10-45

Q

- Quick Form, using, 10-115

R

- Radio buttons, creating, 10-48
- Reading, build output, 12-51

- Rebase section, build output, 12–53
 - Rebuilding, cross reference information, 7–15
 - Record, subsystem table, adding, 20–8
 - Record is changed, 15–21
 - Record Locking
 - in business functions, 15–23
 - understanding, 15–21
 - Record locking, 15–22
 - Records
 - deleting, cursors, 13–19
 - updating, cursors, 13–18
 - Reference list of business functions, displaying, 2–28
 - Related functions, adding, 12–38
 - Related tables and functions, adding, 12–36
 - Replace and preserve with an upgrade, A–3
 - Report, print menu, 17–6
 - Report Data Structure form, defining OneWorld subsystem jobs, 20–5
 - Report data structures
 - modifying, 6–5
 - system generated, 6–1
 - Report Design, OneWorld tools, 1–10
 - Report Design Aid
 - database output, table, 19–1
 - defining OneWorld subsystem jobs, 20–3
 - override, environments, 19–7
 - Report interconnections, creating, 11–85
 - Report modification rules, A–5
 - Report Properties form, defining OneWorld subsystem jobs, 20–4
 - Requirements, currency, internationally, 16–1
 - Resetting
 - cursor, 13–19
 - error, 21–9
 - Find button, 21–10
 - OK button, 21–9
 - Restrict the number of fields
 - business view performance, performance tips, 25–9
 - data structure performance, performance tips, 25–10
 - Reviewing, error output, 12–61
 - Rollback, 15–2
 - Row, exited and changed - asynch, 11–117
 - Row and column text for all applications, changing, 3–21
 - Row is entered, B–48
 - Row is exited, B–48
 - Row is exited and changed (asynchronous portion), B–48
 - Row is exited validation, B–48
 - Runtime
 - Data Dictionary, how used, 3–2
 - processing, 11–9
 - structures, 11–9
 - Runtime data structures and available objects, 11–9
 - Runtime message, glossary text, defining, 21–19
 - Runtime processing
 - all grid recs deleted from DB, 11–30
 - BC assigned database values, 11–19
 - button clicked, 11–26
 - delete grid rec from DB after, 11–30
 - delete grid rec from DB before, 11–29
 - delete grid rec verity after, 11–29
 - delete grid rec verity before, 11–28
 - do in while loop, 11–18
 - grid rec is fetched, 11–19
 - last grid rec has been read, 11–24
 - page-at-a-time processing, 11–18
 - SQL select, 11–18
 - write grid line after, 11–23
 - write grid line before, 11–21
- ## S
- Sample, source code highlights, 22–15
 - Saving and creating, event rules, 11–31
 - Search, BrowsER, 11–126
 - Search and Select
 - forms, 10–12
 - OneWorld tools, Form Design, 1–9
 - Search Combo Box, Event Rules Debugger, 23–10
 - Search/Select form
 - clearing dialog, B–37
 - closing form, B–37
 - data retrieval, B–36
 - default flags, B–35
 - detail data selection and sequencing, B–36
 - dialog initialization, B–35
 - header data retrieval form, B–36
 - menu/toolbar items, B–37
 - add, B–38
 - close, B–37

- copy, B-38
- delete, B-38
- find, B-37
- select, B-37
- user defined items, B-38
- process flow, B-35
 - description, B-35
- Searching
 - batch applications, 7-6
 - business functions, 7-7
 - business views, 7-8
 - data items, 7-4
 - data structures, 7-9
 - event rules, 7-12
 - forms, 7-11
 - interactive applications, 7-5
 - jdeCacheFetchPosition API, 13-19
 - objects, 7-3
 - tables, 7-10
- Section, custom
 - attaching logic, 18-16
 - understanding, 18-15
 - using, 18-17
- Section processing, batch, understanding, 18-9
- Sections
 - group and columnar, processing, 18-9
 - OneWorld, allocates memory, 18-13
 - tabular, processing, 18-11
- Select, menu/toolbar items
 - Find/Browse form, B-5
 - Parent/Child form, B-12
 - Search/Select form, B-37
- Select button, processing, 11-25
- Select Distinct, 5-2
- Select distinct, using, 5-10
- Select/Search form, menu/toolbar items, close, B-37
- Selecting
 - available objects, 6-4
 - business views, 10-20
 - controls, 10-25
 - form type, 10-17
- Send message system function
 - use, 21-26
 - working with, 21-25
- Sensitive controls, currency, showing, 16-6
- Sequencing and data selection
 - performance tips, 25-10
 - processing time, 18-13
- Set focus on grid, B-47
- Set sequence append flag, batch events, 18-31
- Setting
 - automatic error, 21-7
 - build options, 12-49
 - error, 21-7
 - manual error, 21-7
 - business function API, 21-8
 - system function, 21-7
 - paste options, 11-37
- Setting breakpoints, Visual C++ debugger, useful features, 23-22
- Setting up
 - currency, conversion, 16-5
 - JDECACHE, indices, 13-6
- Setting up level breaks, batch application
- performance, performance tips, 25-12
- Show, object ID, BrowsER, 11-125
- Showing, currency, sensitive controls, 16-6
- Simple error message, creating, 21-14
- Single record processing, transaction master business function, implementing, 12-79
- Single-table or multiple-table, business view performance, performance tips, 25-9
- Size, controls using the mouse, 10-27
- Size and place, control using the size command, 10-27
- Size command, size and place a control, 10-27
- Size of alias, for use in an RPG program, 3-13
- Smart Field, creating a data item, C-11
- Smart field triggers, attaching, 3-38
- Smart Fields, C-1
 - creating, C-7
 - understanding, C-3
- Source code highlights, sample, 22-15
- Source file, function name, cache business function description, 13-35
- Special logic/processing required
 - begin document, 12-68, 12-70
 - cancel document (optional), 12-75
 - clear cache, 12-74
 - edit document, 12-72
 - end document, 12-74
- Specification file corruption, index limitations for various databases, performance tips, 25-8
- SQL log, tracing, 23-28
- SQL select, Runtime processing, 11-18
- SQLSERVER, index limitations for various databases, performance tips, 25-6
- Standard, JDEBASE API categories, 12-27

Standard naming convention, business functions, cache business function source name, 13–35

Standard parameters, 12–83

Standard processing, media objects, using, 10–118

Standards

- cache action code, 13–37
 - CACHE_ADD, 13–37
 - CACHE_ADD_UPDATE, 13–38
 - CACHE_CLOSE_CURSOR, 13–39
 - CACHE_DELETE, 13–38
 - CACHE_GET, 13–37
 - CACHE_GET_NEXT, 13–39
 - CACHE_TERMINATE, 13–39
 - CACHE_TERMINATE_ALL, 13–39
 - CACHE_UPDATE, 13–38
- cache programming, 13–36
 - general, 13–36

Standards and portability, understanding, 12–25

Static text, controls

- creating, 10–52
- disconnecting, 10–60

Step command, Visual C++ debugger, useful features, 23–21

Step into command, Visual C++ debugger, useful features, 23–21

Steps, database communication, understanding, 12–28

Stop section processing, batch events, 18–32

Store, objects, 1–7

Storing, Data Dictionary, 3–3

- data items, 3–3

Strategies, debugging, 23–25

Structure

- business function source files, 12–20
- cache, 12–75
 - detail, 12–77
 - header, 12–76

Structures, runtime, 11–9

Styles trigger, overriding, 10–105

Subcategories, creating in menus, 10–40

Subheading, 17–21

Subsystem jobs

- defining, 20–3
- defining for OneWorld, 20–3
- OneWorld, batch processes, 20–1

Subsystem table

- adding, record, 20–8

- API record, adding, 20–7

Summary section, build output, 12–54

Support, JDECACHE multiple cursor, 13–20

System function

- event, attaching, 11–61
- manual error, setting, 21–7
- tracing, 23–29

System functions within batch events, 18–31

System generated data structures, 6–1

- form, 6–1
- report, 6–1

T

Tab sequence, changing, 10–92

Table

- choosing, business view, 5–8
- event rules, 11–4
- Form Information Table -F9865, 2–5
- Next Numbers - F0002, 3–36
- Object Librarian Detail Table -F9861, 2–3
- Object Librarian Function Detail -F9862, 2–3
- Object Librarian Master Table -F9860, 2–2
- Object Librarian-Business Function Object Relationships -F9863, 2–4
- Report Design Aid, database output, 19–1
- subsystem
 - adding a record, 20–8
 - adding and API record, 20–7

Table Design

- OneWorld tools, 1–9, 4–1
- working with, 4–9

Table design performance, performance tips, 25–5

- index considerations, 25–5
- World coexistence considerations, 25–6

Table event rules, 11–107

- creating, 11–107

Table I/O

- available operations, 11–97
- event rule, creating, 11–98
- event rule variable, 11–58
- Event Rules Design, 11–97
- valid mapping operators, 11–98

Table I/O objects, performance tips, 25–8

Table I/O versus joined business view, business view performance, performance tips, 25–10

- Table join, 5–1
 - creating, 5–14
- Table specification modification rules, A–6
- Table union, 5–2
 - creating, 5–16
- Tables
 - adding, 4–3
 - J.D. Edwards naming standards, 4–4
 - choosing data items, 4–10
 - copying, 4–18
 - defined, 4–1
 - generating, 4–13
 - menu design, 17–2
 - naming conventions, 2–12
 - previewing, 4–17
 - searching, 7–10
 - viewing data in, 4–21
- tables, viewing data within, 4–21
- Tabular sections, processing, 18–11
- Template
 - business function documentation, 12–90
 - data structure documentation, 12–94
 - processing options, 14–2
- Templates
 - category codes, 2–17
 - function use, 2–17
 - name, 2–17
- Terminating
 - cache, 13–36
 - work center processes, 22–23
- Testing, forms, 10–127
- Text functions, advanced functions, expression manager, 11–94
- Text substitution, error message, creating, 21–16
- Text substitution error messages, 22–5
- Text translation, processing option template, 14–14
- Text Variables, Creating, 10–111
- Things to look for, applications, performance tips, 25–3
- Third-party software, imaging, 10–117
- Threaded events, asynchronous events, 11–116
- Tracing
 - debugging, 23–29
 - turn on, 23–29
 - SQL log, 23–28
 - system function, 23–29
- Tracking objects, OneWorld tools, Object Librarian, 2–2
- Transaction boundaries, 15–3
- Transaction boundary, extending, 15–9
 - between forms, 15–9
 - business functions, 15–13
- Table I/O, 15–16
- Transaction master business function, Components, 12–67
- Transaction master business function modules, naming, 12–66
 - begin document, 12–67
 - cancel document (optional), 12–75
 - clear cache, 12–74
 - Edit document, 12–72
 - edit line, 12–70
 - end document, 12–73
- Transaction master business functions, 12–65
 - building, 12–78
 - implementing, 12–79
 - document processing, 12–80
 - single record processing, 12–79
- Transaction Processing, 15–1
 - and business functions, 15–5
 - commits, 15–1
 - commit (manual commit mode), 15–2
 - rollback, 15–2
 - two phase, 15–2
 - data interdependence, 15–3
 - database locking, 15–21
 - record is changed, 15–21
 - record locking, 15–22
 - defining for a form, 15–8
 - defining for a report, 15–19
 - in remote business functions, 15–6
 - OneWorld, 15–3
 - scenarios, 15–4
 - system functions, 15–6
 - transaction boundaries, 15–3
 - understanding, 15–3
 - working with, 15–7
- Tree
 - controls, 10–68
 - loading nodes, 10–83
 - node is selected, 10–86
- Tree node expand, data retrieval, Parent/Child form, B–11
- Trig functions, advanced functions, expression manager, 11–94
- Triggers
 - attaching
 - default, 3–23

- default value, 3–25
 - display rule, 3–31
 - edit rule, 3–28
 - next number, 3–36
 - smart field, 3–38
 - user defined codes triggers, 3–33
 - visual assist, 3–26
 - Data Dictionary performance, performance tips, 25–4
 - default, understanding, 3–4
 - override
 - decimal, 10–106
 - default value, 10–99
 - edit rule, 10–102
 - formatting, 10–101
 - next number, 10–104
 - previously defined, 10–97
 - styles, 10–105
 - user defined codes, 10–103
 - visual assist, 10–100
 - Triggers option, build, 16–3
 - Turn on, debugging, tracing, 23–29
 - Two Phase Commit, 15–2
 - Type definitions, displaying, 6–7
 - Types
 - Data Dictionary
 - error messages, 3–4
 - glossary items, 3–4
 - JDECACHE cursors APIs, 13–17
 - Typical, event flow, Find/Browse form, 11–14
 - Typical users and hook up, begin document, 12–71
- ## U
- UDC edit controls
 - creating, 10–58
 - defining options, 10–90
 - Understanding level-break processing, 18–19
 - Understanding
 - applications, 9–3
 - batch error messages, 22–3
 - breakpoints, debugging, 23–9
 - business function builder form, 12–45
 - C business functions, 12–15
 - controls, 11–2
 - custom section, 18–15
 - database communications steps, 12–28
 - error handling, 21–5
 - event rules, 11–3
 - event rules buttons, 11–4
 - events, 11–2
 - form processing, 11–2
 - form types, 10–5
 - java applications, 9–4
 - level-break message, 22–6
 - modification rules, report specifications , A–5
 - portability and standards, 12–25
 - smart fields, C–3
 - standards and portability, 12–25
 - Transaction Processing, 15–3
 - web applications, 9–4
 - Understanding Applications, 1–4
 - Understanding batch events, 18–23
 - Understanding default triggers, 3–4
 - Understanding how OneWorld stores objects, 1–7
 - Understanding how to build applications, 1–5
 - Understanding Objects, 1–3
 - Understanding OneWorld tools, 1–8
 - Understanding section processing, 18–9
 - Understanding the development cycle, 1–6
 - Unexpectedly ending, program, 23–25
 - Unions, business view performance, performance tips, 25–9
 - Universal Table Browser, working with, 4–21
 - Universal table browser
 - Formatted data, example, 4–25, 4–26
 - non-formatted data, example, 4–24
 - work with, 4–21, 4–25
 - Updating
 - Data Dictionary items for languages, 3–20
 - glossary, 3–40
 - records, cursor, 13–18
 - Upgrade
 - preserve and replace, A–3
 - replace and preserve, A–3
 - Upgrade preserve and replace, A–3
 - Use, send message system function, 21–26
 - Useful features, Visual C ++ debugger, 23–21
 - go command, 23–21
 - locals window, 23–22
 - setting breakpoints, 23–22
 - step command, 23–21
 - step into command, 23–21
 - Using watch, 23–22
 - User defined codes trigger, overriding, 10–103

- User defined codes triggers, attaching, 3–33
- User defined items, menu/toolbar items
 - Find/Browse form, B–6
 - Fix/Inspect form, B–18
 - Header Detail form, B–26
 - Headerless Detail form, B–33
 - Parent/Child form, B–13
 - Search/Select form, B–38
- User generated data structures, 6–1
 - business function, 6–2
 - media objects, 6–1
 - processing options, 6–1
- Using
 - custom section, 18–17
 - database APIs, 12–25
 - jdeCacheInit/jdeCacheTerminate rule, 13–12
 - same cache in multiple business functions or forms, 13–13
 - select distinct, 5–10
 - utility programs, 12–51
- Using Data Dictionary, 3–7
- Using Quick Form, 10–115
- Using standard processing, media objects, 10–118
- Using watch, Visual C++ debugger, useful features, 23–22
- Utility programs, using, 12–51

V

- Valid mapping operators, Table I/O, 11–98
- Validating, event rules, 11–42
- Validation, Data Dictionary performance, performance tips, 25–5
- Value, assign, 11–89
- Variable
 - inspecting or modifying, debugging, 23–12
 - modifying or inspecting, debugging, 23–12
- Variable Selection and Display Control, Event Rules Debugger, 23–9
- Variables
 - Grid, 11–53
 - text, creating, 10–111
- Version modification rules, A–10
- Viewing
 - documentation
 - business function - values to pass, 12–99

- business function document viewer, 12–100
 - business function search, 12–98
 - field relationships, 7–14
- Visual assist trigger, overriding, 10–100
- Visual assist triggers, attaching, 3–26
- Visual C++ debugger
 - useful features, 23–21
 - go command, 23–21
 - locals windows, 23–22
 - setting breakpoints, 23–22
 - step command, 23–21
 - step into command, 23–21
 - using watch, 23–22
 - working with, 23–21
- Visual C++ debugging, customizing the environment, 23–23

W

- Web
 - applications, 9–4, 24–4, 24–7
 - generating
 - all applications, 24–15
 - applications, 24–14
- Web applications, generating, forms, 24–9
- Web-based solutions, developing, 24–3
- What an upgrade preserves and replaces, A–3
- While statements, event rule logic, 11–46
- Windows application, add to a menu, 17–18
- Work Center API, error message, 22–7
- Work Center APIs, calling, 22–20
- Work center initialization API, calling, object librarian, 22–18
- Work center processes, terminating, 22–23
- Work With Object Librarian Form, 12–42
- Work With Object Librarian form, 2–18
- Workflow, message types, 21–2
- Working with
 - asynchronous processing, 11–115
 - BrowsER, 11–121
 - BrowsER options, 11–124
 - business function builder, 12–45
 - business function source librarian, 2–29
 - business functions, 12–35
 - Business View Design, 5–7
 - checkout log, 8–3
 - controls, 10–43

- cross reference facility, 7–3
- currency, 16–3
- Event Rule Design, 11–31
- event rule variables, 11–53
- Event Rules, debugging, 23–5
- event rules, currency processing, 16–4
- if and while statements, 11–45
- interconnect data structures, 6–3
- JDECACHE, 13–5
- messages, 21–13
- Object Librarian, 2–7
- Object Librarian Codes, 2–23
- send message system function, 21–25
- Table Design, 4–9
- transaction processing, 15–7
- Universal Table Browser, 4–21
- Visual C++ debugger, 23–21
- World coexistence considerations, table design
- performance, performance tips, 25–6
- WorldVision application, add to a menu, 17–17
- Write grid line after, runtime processing, 11–23
- Write grid line before, runtime processing,
11–21

