

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



- Using Visual ER Compare

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

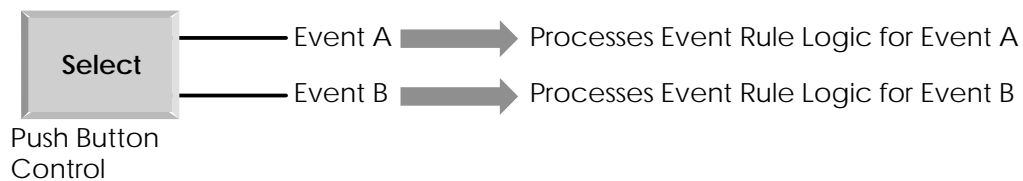
Form processing refers to the business logic associated with each form. By default, each type of OneWorld form automatically includes basic processing for various events. Then, using Event Rules Design, you can build additional logic.

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

Grid-Level Events

Fix/Inspect

Header Detail and Headerless Detail

Form-Level Events

Grid-Level Events

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 Data Structures

Runtime data 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.

Runtime data structures are created dynamically by the runtime system. 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 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

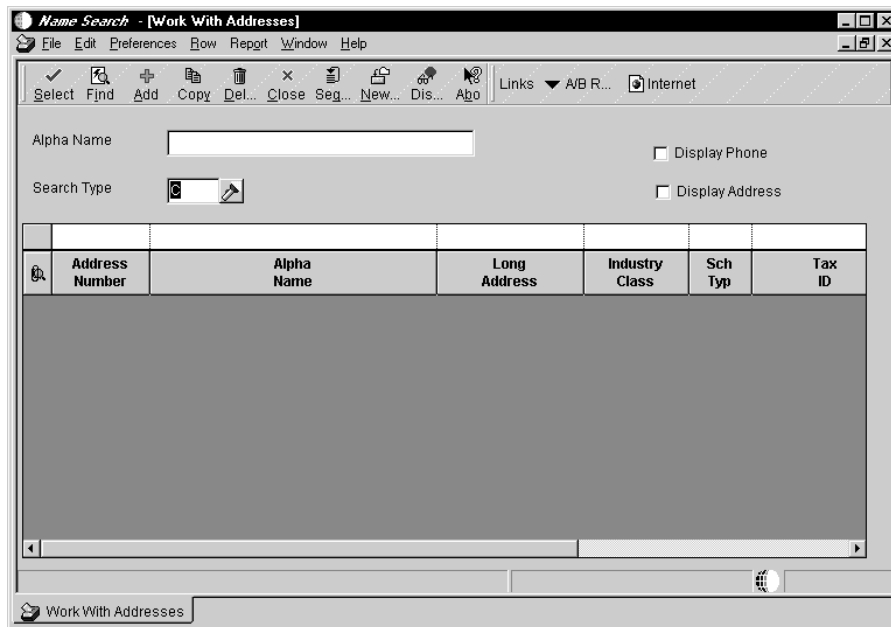
Each form type has its own properties and its own event flow. Processing occurs whether or not you add event rule logic. However, the engine pauses at specific events, such as *Dialog Is Initialized*, so you can add logic or manipulate values. How events execute is based on event rule logic and user interaction.

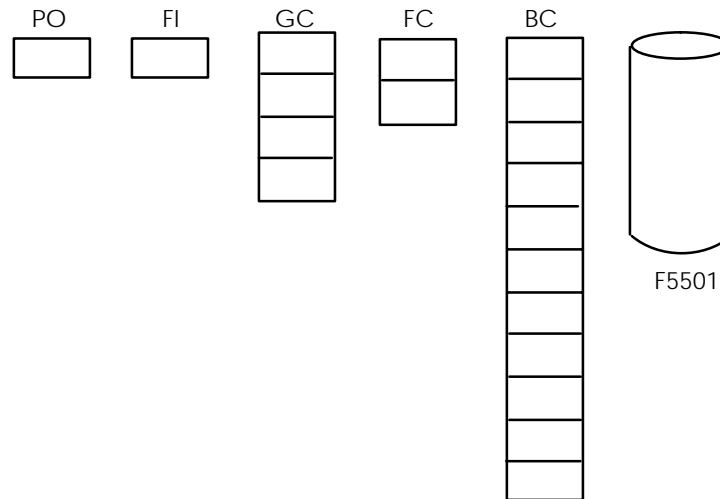
Some of these events occur on each form, although the order and number of events varies depending on the form type. For example, *Dialog Is Initialized* is a good place to add logic on all forms. *Post Dialog is Initialized* is a good place to add logic on a Fix/Inspect form, but not on a Find/Browse 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 updatable 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.





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

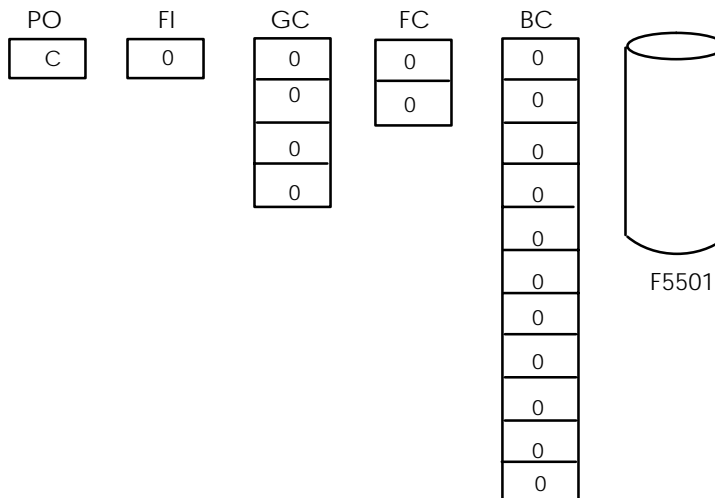
FC=Any FI values passed

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)

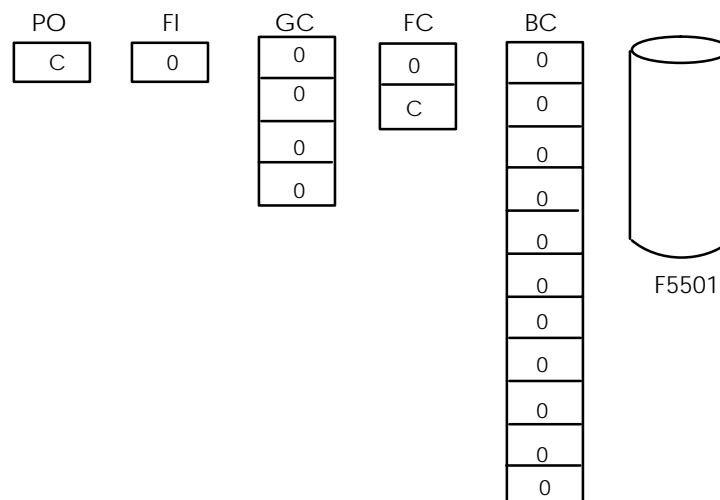
GC=0 (or values already passed in)

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.

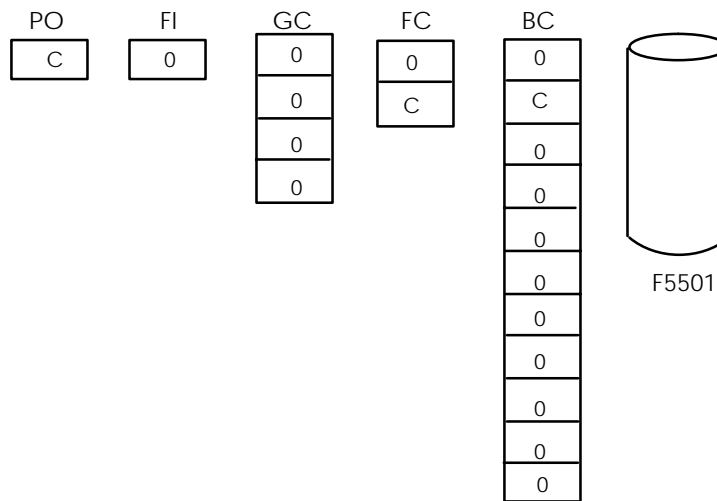
At this point, the user must click the Find button to populate the grid. (The J.D. Edwards standard is to disable autofind.) The SQL SELECT is now built.

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.

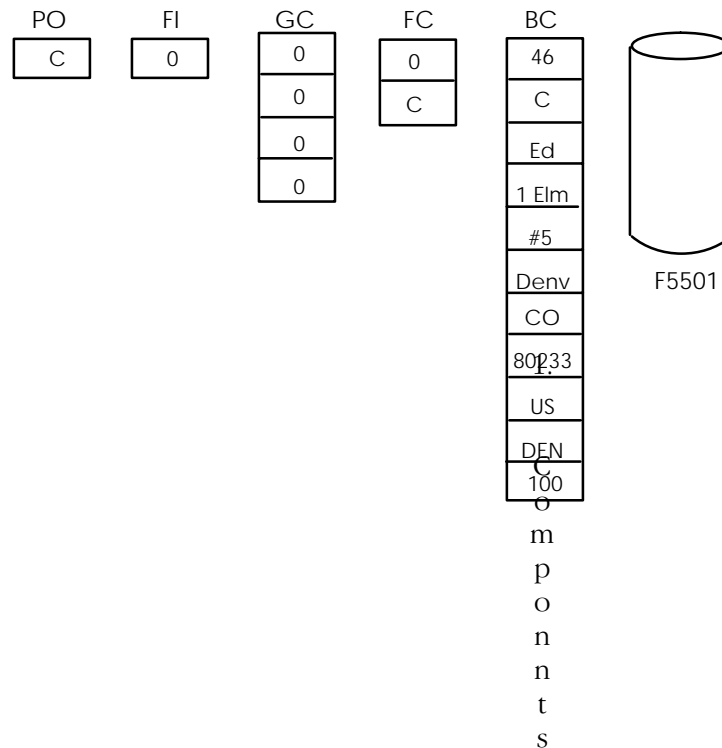
If page-at-a-time processing is on, event rule logic attached to the *Last Grid Rec Has Been Fetch* event will not be executed if the user has not clicked the down arrow key on the grid scroll bar enough to fetch the last record.

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)

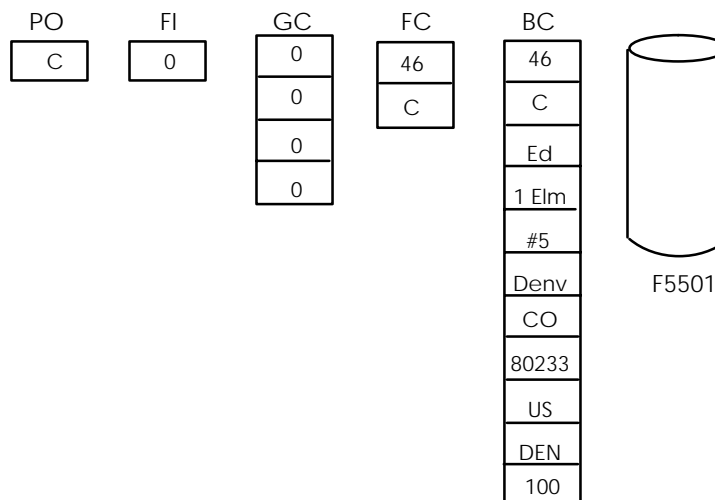
FC=Values from the database (if database fields)

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.

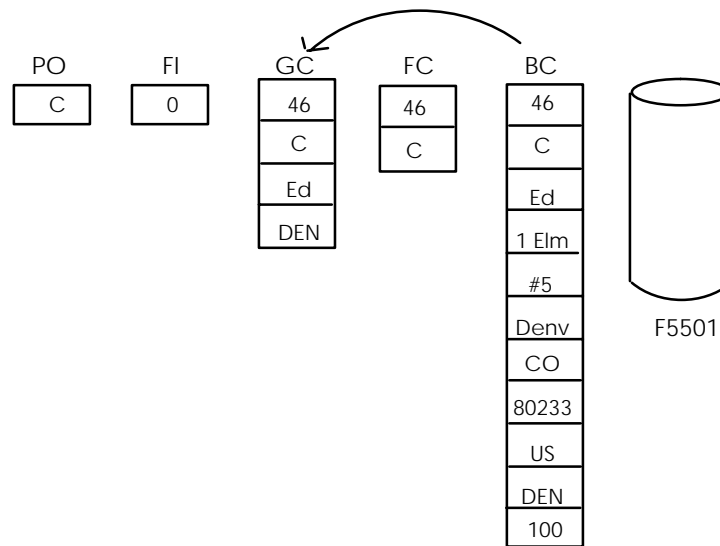


The *Grid Rec Is Fetched* event is commonly used to:

- 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)

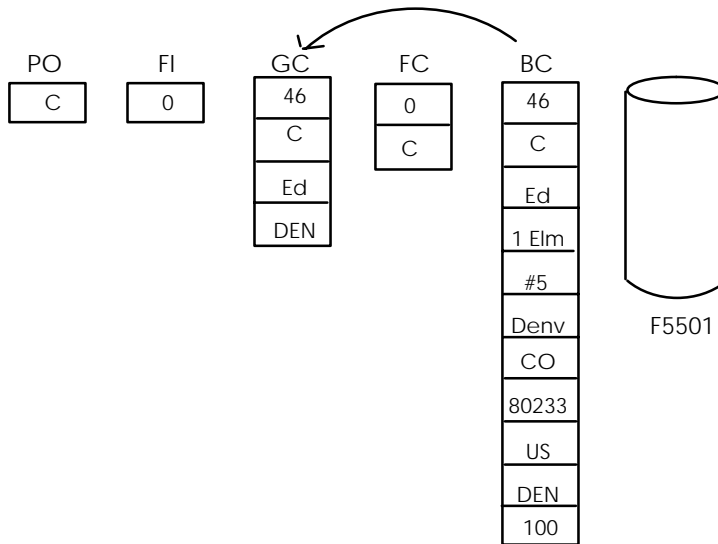
FC=Values from the database (if database fields)

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.

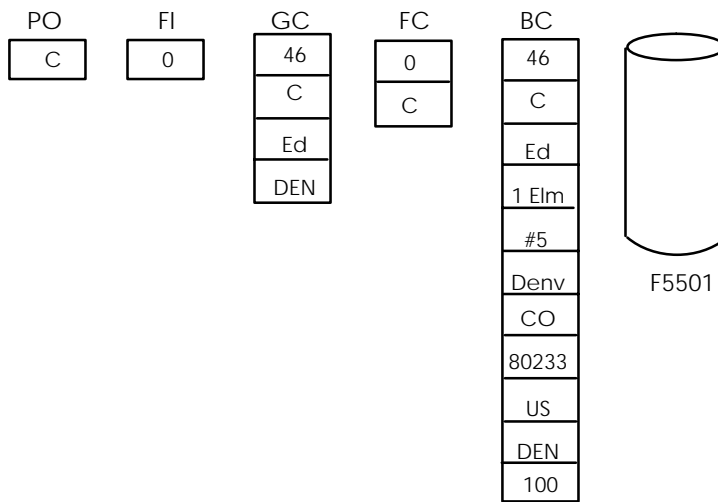


The *Write Grid Line Before* event is commonly used to:

- 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)

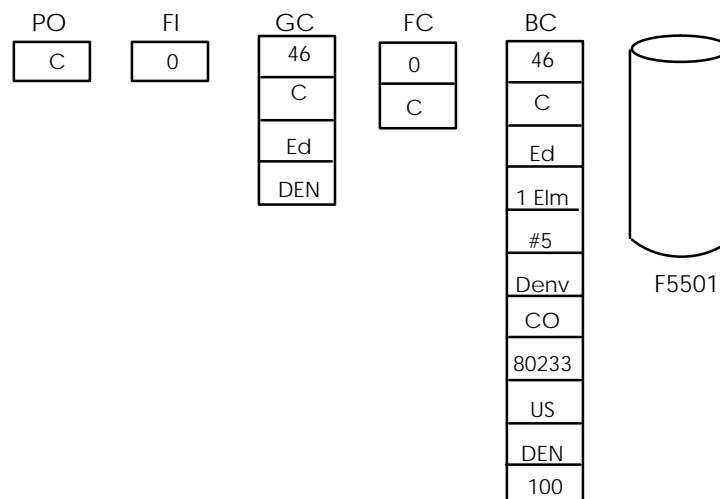
GC=Values from the database (from the first record read)

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.



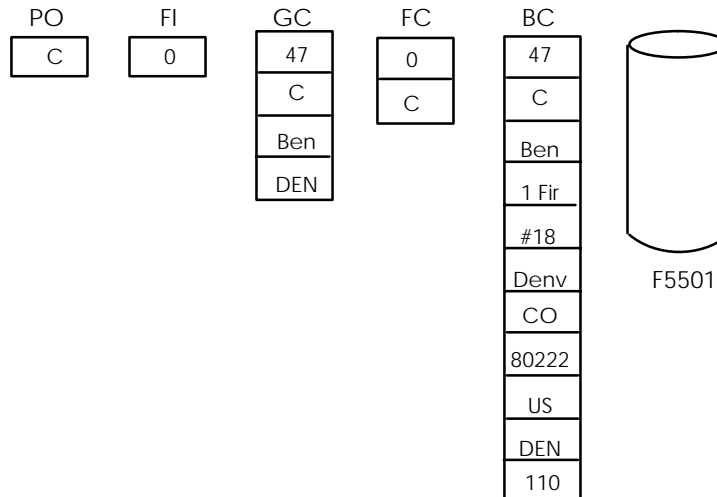
The *Write Grid Line After* event is commonly used to add logic after the user sees a row on the form.

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)

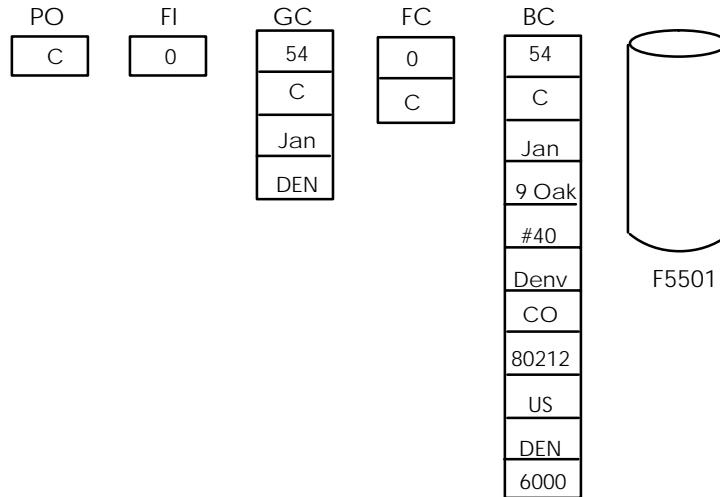
GC=Values from the database (from the last record read)

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.



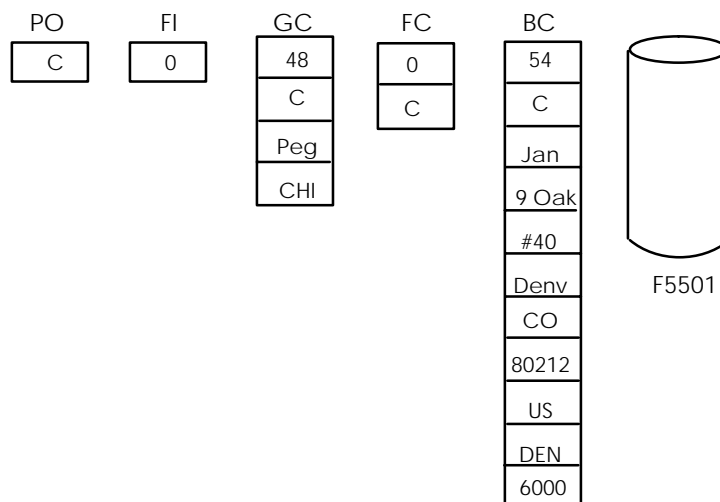
The *Last Grid Rec Has Been Read* event is commonly used to:

- 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)

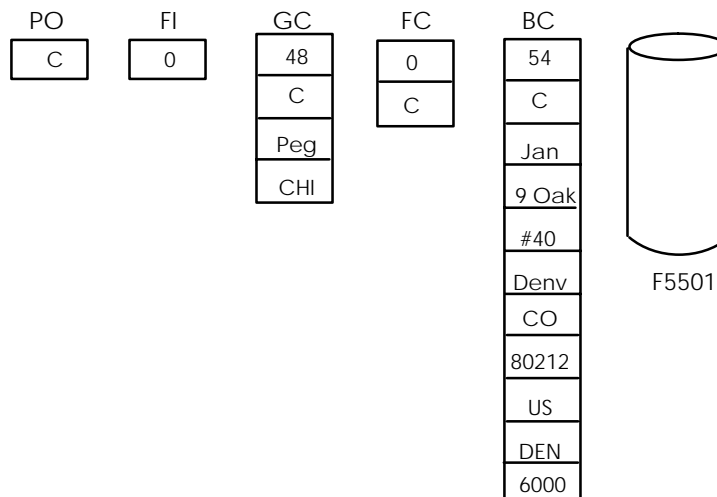
GC=Values from the selected grid row

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)

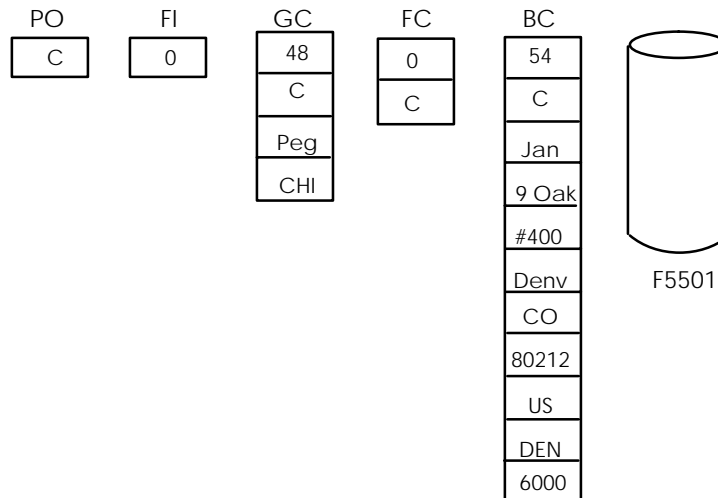
GC=Values from the selected grid row

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)

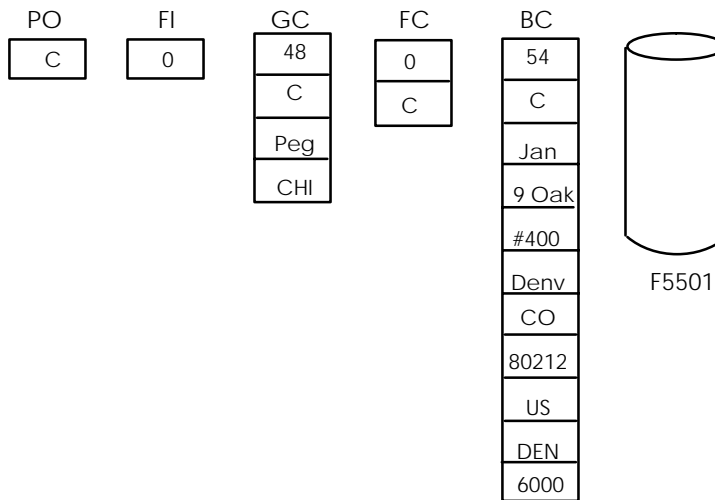
FC=Values from the database (if database field)

GC=Values from the selected grid row

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

PO=Values passed from processing options

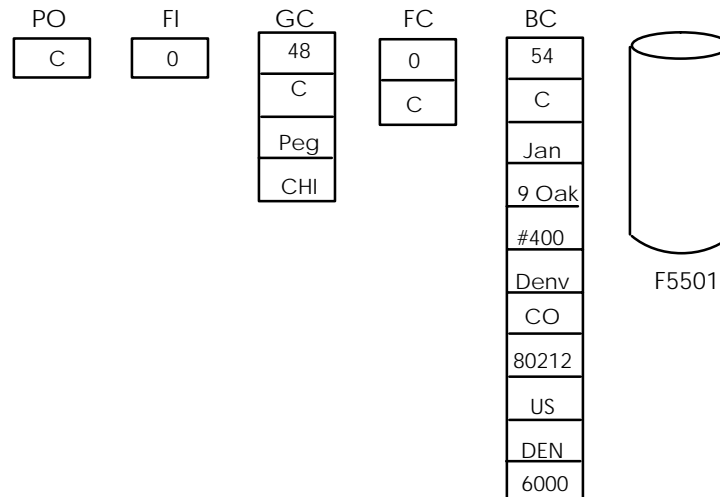
The following illustration represents what is in the runtime structures just before *Button Clicked* is run.



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
- Finding event rules
- Cutting, copying, and pasting 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?

See Also

- Understanding the HTML Client* for information about turning on and off HTML post (refresh) for an event

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.

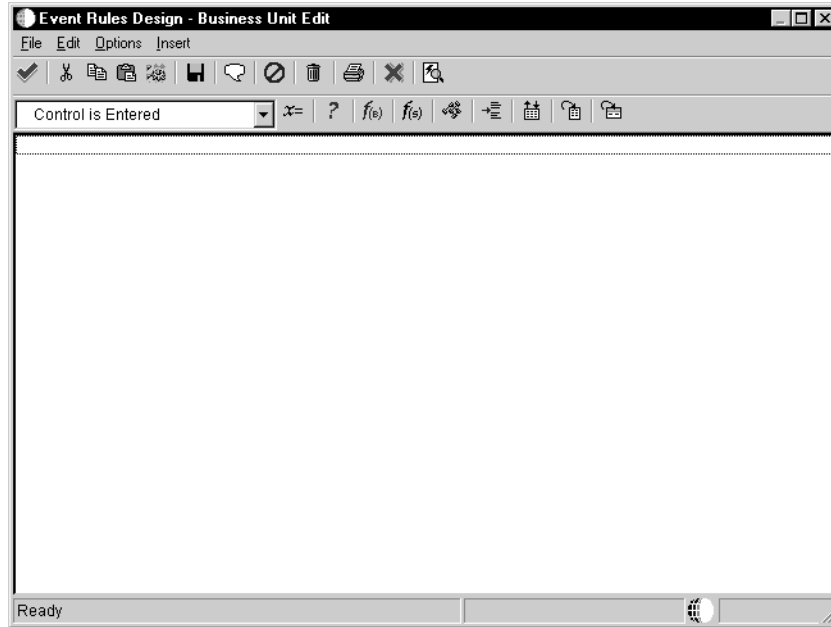
Create event rules by clicking the buttons on the toolbar in Event Rules Design. Depending on the button that you click, a different work area displays for creating and manipulating the event rule line by line.

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

1. On Forms Design, 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	<p>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.</p> <p>Disabled event rules are marked with a red ! in the event list.</p>
Repeat Business Rules for Grid – ER Dsn	<p>Applies an event rule to each of the grid selections. This check box is available only for grid controls.</p> <p>Note: Not all grids allow multiple selections. The developer must specify the Multiple Selection option in Grid Properties form.</p>
Save	<p>Saves all rules for all events associated with the selected control.</p> <p>Note: The OK button also performs a save before exiting.</p>
Comment - Event Rules Design	<p>Inserts a comment into the event rule. Comments do not affect logic. Use comments to document the business rules.</p>
Enable/Disable - Event Rules Design	<p>Enables or disables a single line of the event rule. Disabled event rule lines are marked with a red exclamation (!).</p>
Delete	<p>Destroys an object or record.</p> <p>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.</p>

Finding Event Rules

You can search for strings in event rules.

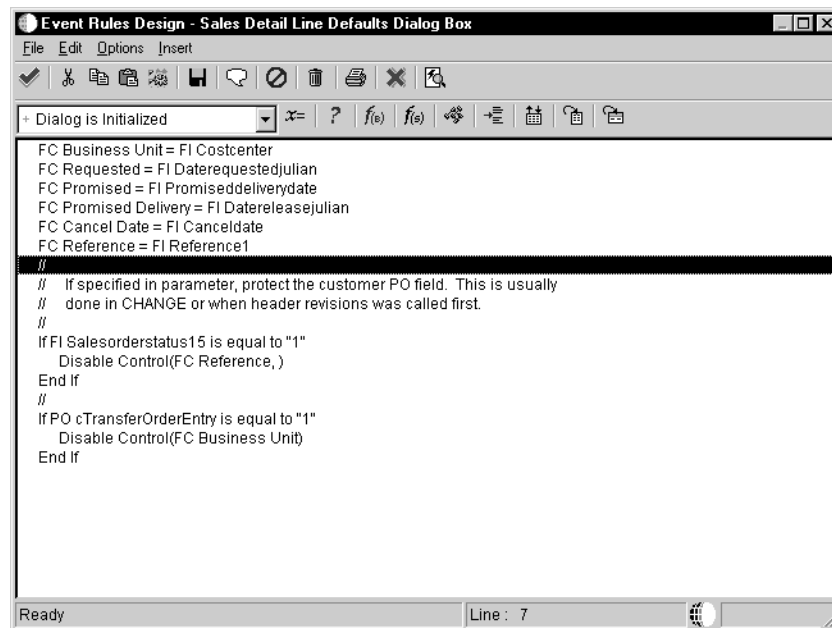
► To find a string of text in an event rule

1. From Event Rules Design, choose Find from the Edit menu.
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**

1. On Event Rules Design, select one or more lines of event rules.
2. Click the Cut button.

▶ **To copy event rules**

1. On Event Rules Design, select one or more lines of event rules.
2. Click the Copy button.

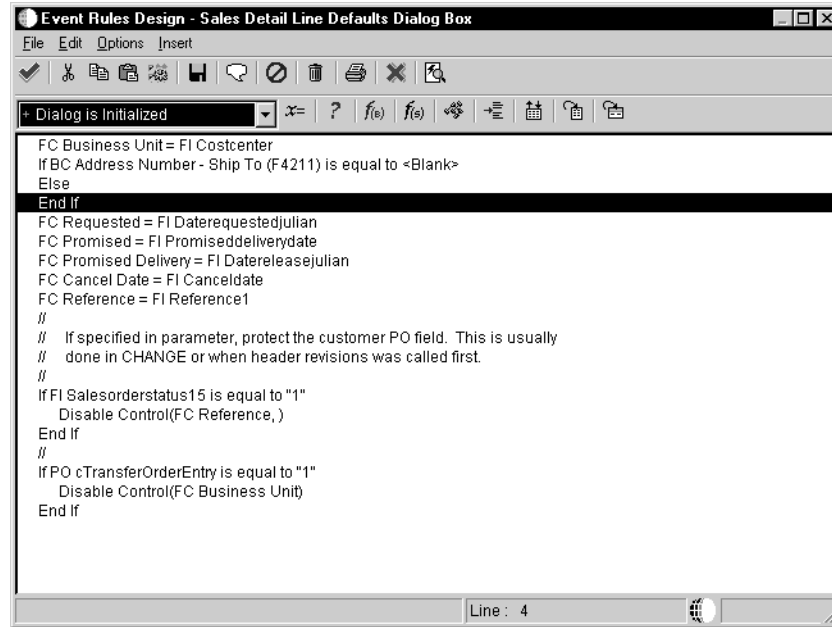
When you copy event rules, the selected lines of event rules are copied from the source and stored on the clipboard.

▶ **To paste event rules**

1. On Event Rules Design, 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.

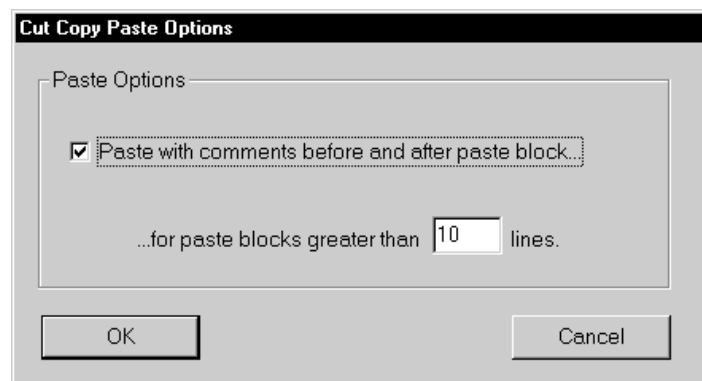


For criteria statements, the paste operation adds whatever is necessary to maintain a clean logical structure. For example, if you paste an If statement and there is no EndIf statement, the paste operation will add a matching EndIf statement to make the logic complete.

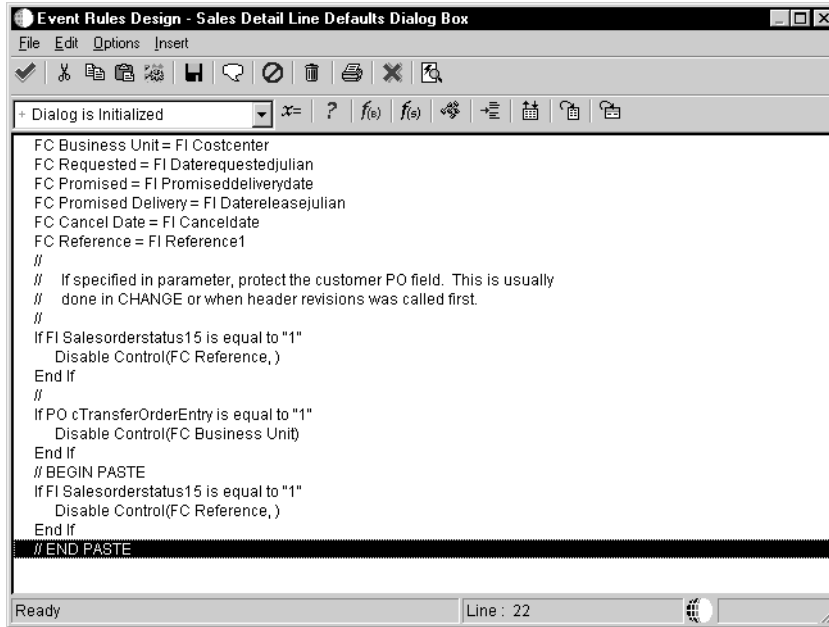
You can set paste options to display comments before and after a block of pasted event rules.

► 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.

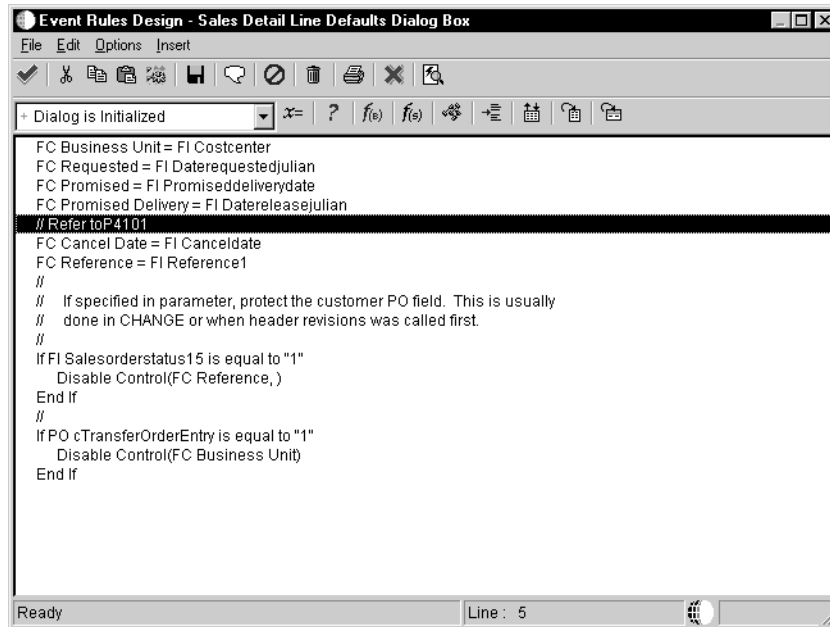
► To add a comment line to event rule

1. On Event Rules Design, 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

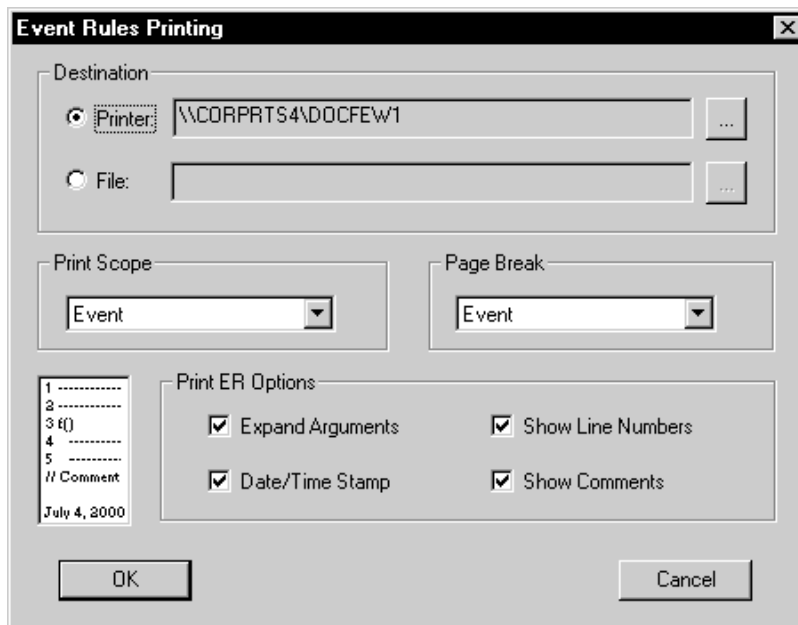
► To print event rule code

1. On Form Design, 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	<p>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:</p> <ul style="list-style-type: none"> • 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.

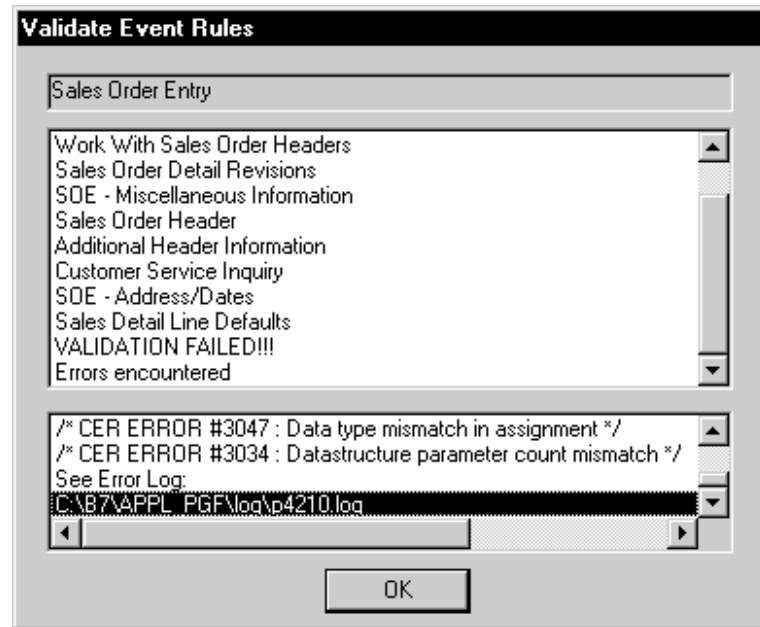
- If you select Validate Event Rules Now, event rules will be validated immediately.

► To validate event rules

1. On Forms Design, from the Application menu select:
 - Validate Event Rules Now

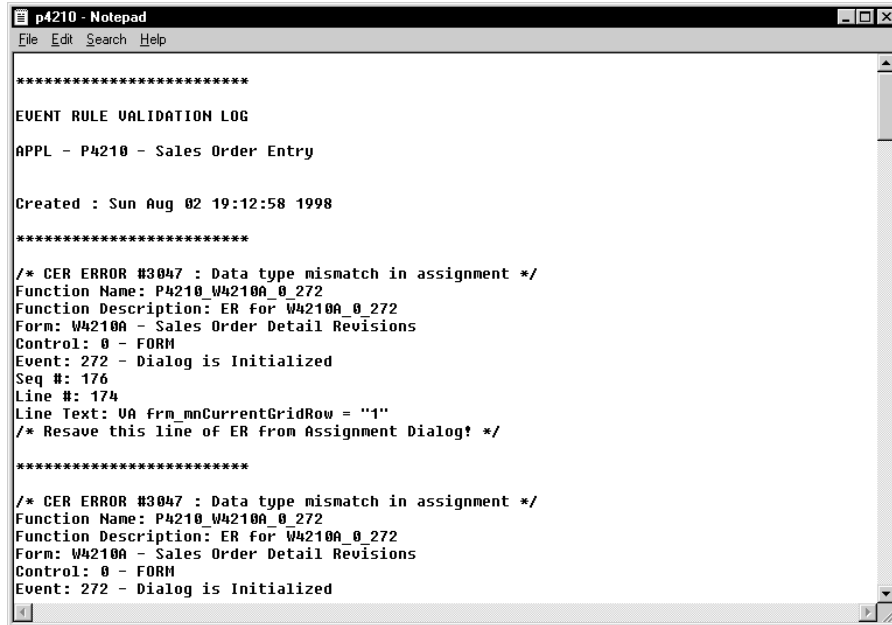
The Validate Event Rules form displays messages as it processes. When validation has successfully completed a generation successful message appears.

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

*****

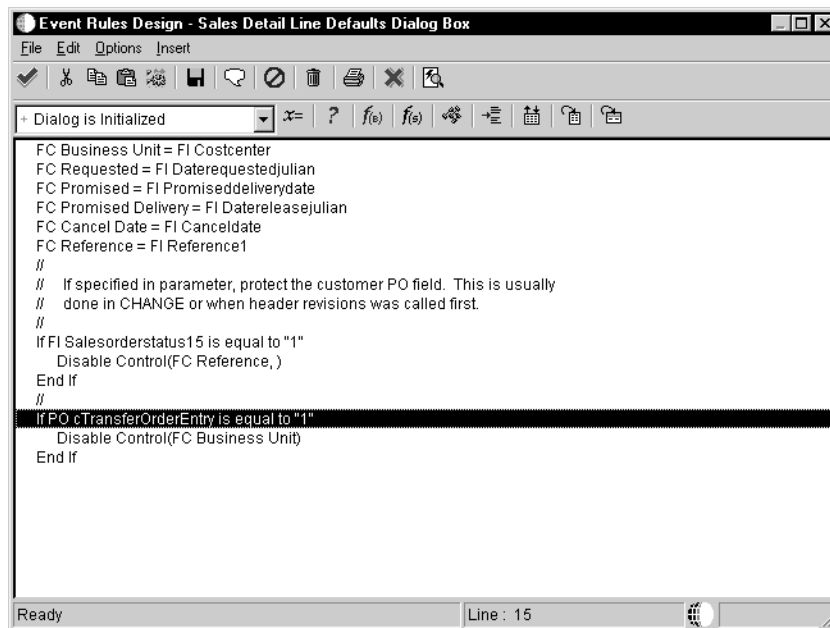
/* CER ERROR #3047 : Data type mismatch in assignment */
Function Name: P4210_W4210A_0_272
Function Description: ER for W4210A_0_272
Form: W4210A - Sales Order Detail Revisions
Control: 0 - FORM
Event: 272 - Dialog is Initialized
Seq #: 176
Line #: 174
Line Text: VA frm_mnCurrentGridRow = "1"
/* Resave 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:

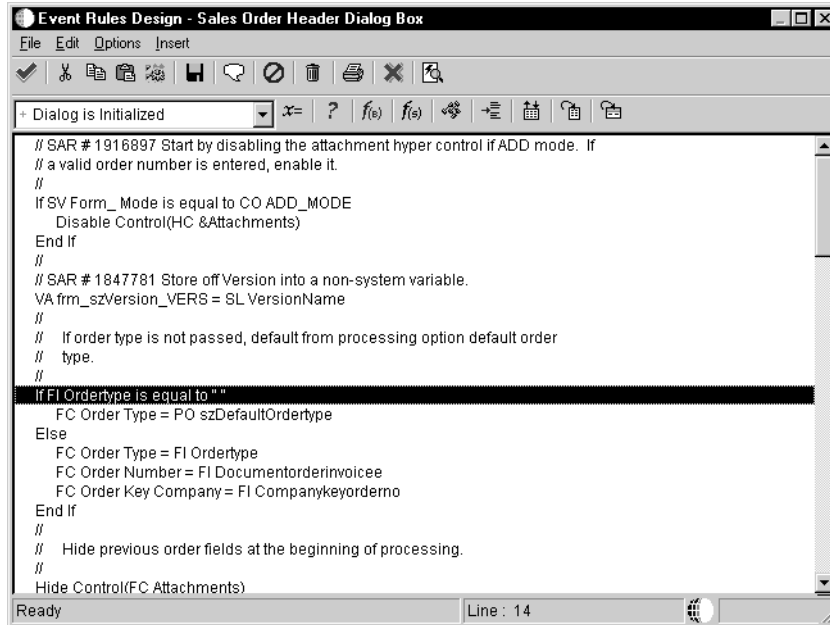
```
IF condition X is true
  Do Y
ELSE
  Do Z
ENDIF
```

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. 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. 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.

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 or edit the event rule to eliminate the errors.

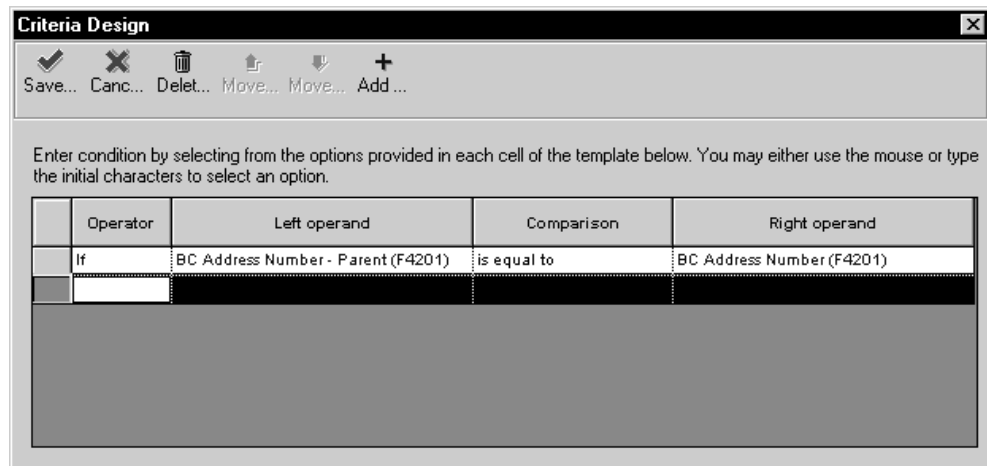
“Is Equal To” and “Is Not Equal To” are the only valid logical operators when using range and list values. The criteria design tool protects you against using the range and list literals incorrectly by ensuring that the comparison is either equal to or not equal to. If you choose Range or List and your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

To change 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.

To delete a statement, select the line of the statement, and then 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.

► To create an If or a While statement

1. On Event Rules Design, select an event in the Event Rules Design window.
2. Click the If/While button.



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

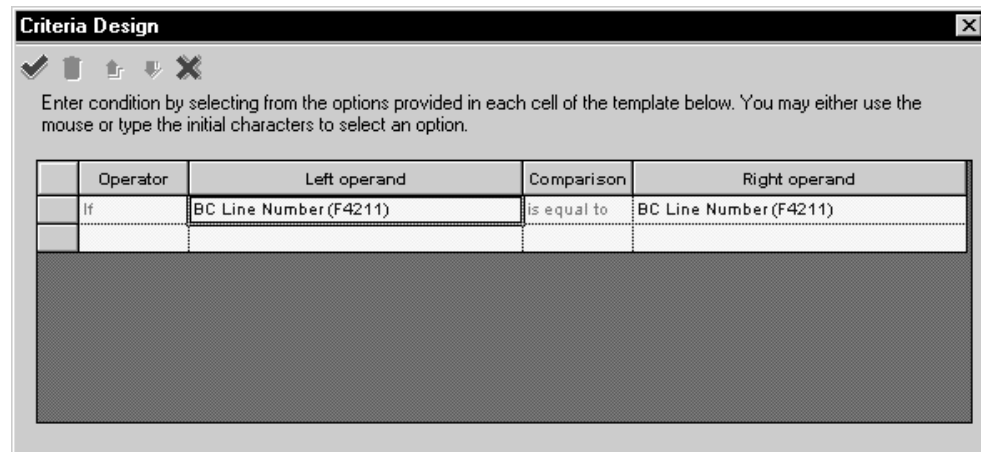
5. Choose a Comparison from the list of logical operators.

- 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.



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.

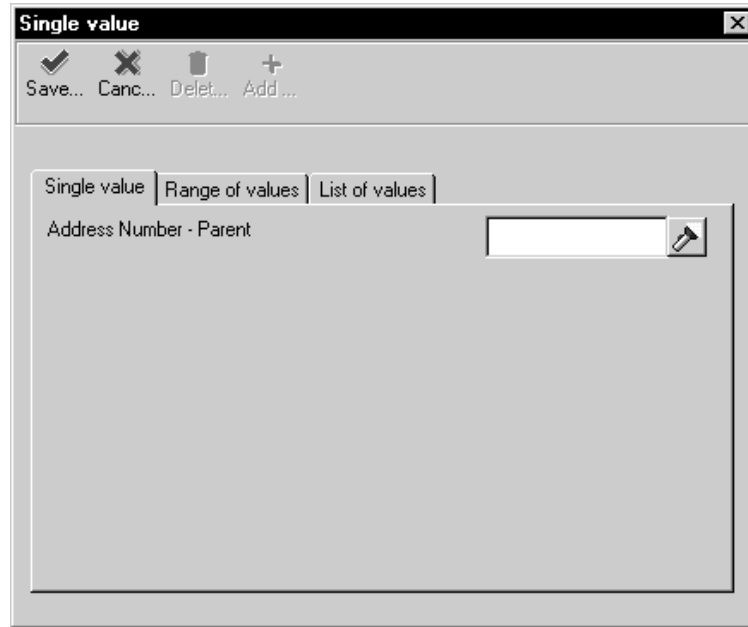
You can select the And or Or option to continue your logic to create complex if statements.

Defining Literal Values in Event Rule Logic

You can define literal values in event rules.

► To define literal values

1. On Criteria Design, from the Right operand list, double-click 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.

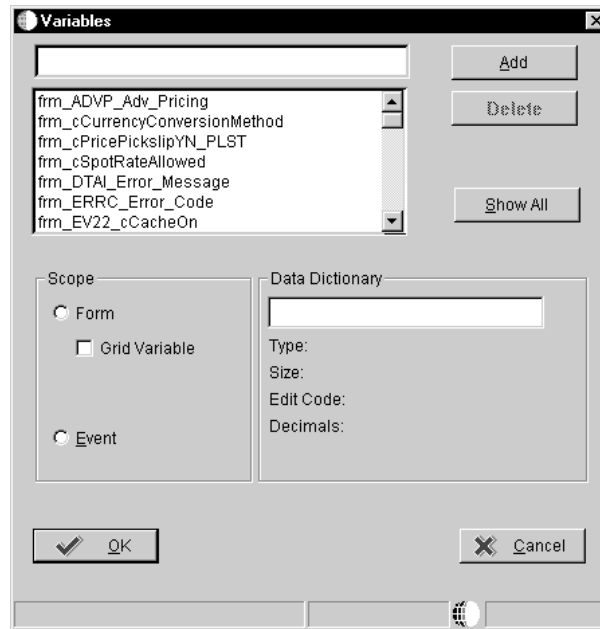
Variables are automatically assigned one of the following prefixes based on the specified scope:

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

To create an event rule variable

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

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



2. Complete the variable naming field.

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

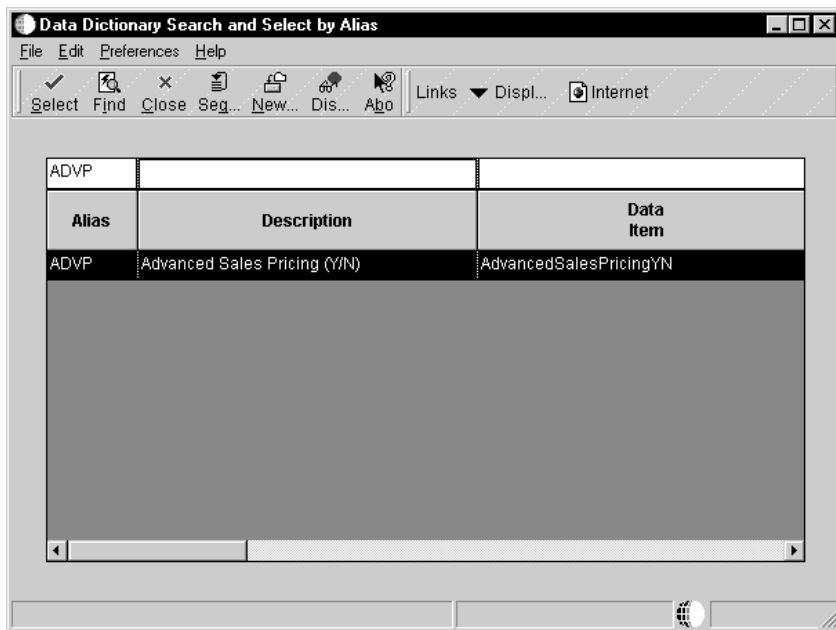
id - Pointer

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.

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.



5. Click the Data Dictionary visual assist to browse data dictionary items.

Alias	Description	Data Item
AE	Amount - Extended Price Prior Month	AmtExtendPricePrior
AE01	Amount - Extended Price Prior Month 01	AmtExtendPricePrior1
AE02	Amount - Extended Price Prior Month 02	AmtExtendPricePrior2
AE03	Amount - Extended Price Prior Month 03	AmtExtendPricePrior3
AE04	Amount - Extended Price Prior Month 04	AmtExtendPricePrior4
AE05	Amount - Extended Price Prior Month 05	AmtExtendPricePrior5
AE06	Amount - Extended Price Prior Month 06	AmtExtendPricePrior6
AE07	Amount - Extended Price Prior Month 07	AmtExtendPricePrior7
AE08	Amount - Extended Price Prior Month 08	AmtExtendPricePrior8
AE09	Amount - Extended Price Prior Month 09	AmtExtendPricePrior9
AE10	Amount - Extended Price Prior Month 10	AmtExtendPricePrior10

Variables

frm_ADVP_Adv_Pricing
 frm_cCurrencyConversionMethod
 frm_cPricePickslipYN_PLST
 frm_cSpotRateAllowed
 frm_DTAL_Error_Message
 frm_ERRC_Error_Code
 frm_EV22_cCacheOn

Buttons: Add, Delete, Show All

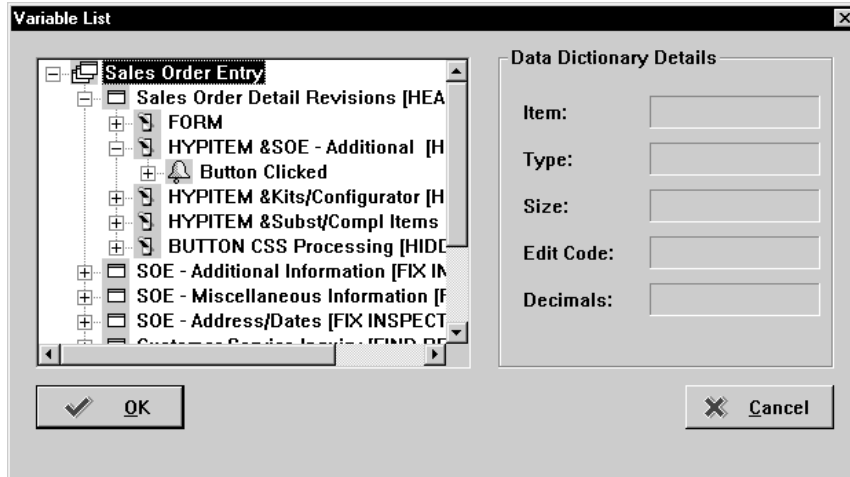
Scope: Form, Grid Variable, Event

Data Dictionary: ADVP

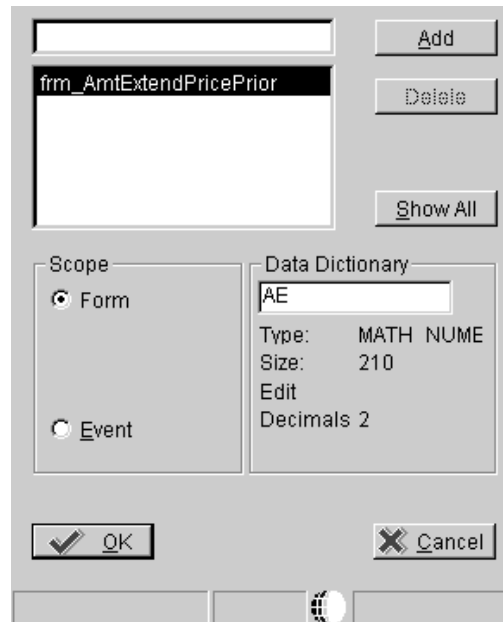
Type:
 Size:
 Edit Code:
 Decimals:

Buttons: OK, Cancel

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.

Field	Explanation
Scope - Variables	<p>Determines how the variable can be used. For example, you can reference:</p> <ul style="list-style-type: none"> • A report variable anywhere in the report • An event variable only within the event where it was created <p>Different scope options are available for interactive and batch applications.</p>

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_LineNumberCounter_LNID
```

```
VA frm_LineNumberCounter_LNID = BC LineNumber
```

```
End If
```

4. On the Add Last Entry Row to Grid event, increment the LineNumber and assign the new value to the next available line.

```
VA frm_LineNumberCounter_LNID = VA  
frm_LineNumberCounter_LNID + 1
```

```
GC LineNumber = VA frm_LineNumberCounter_LNID
```


Attaching Functions

You can attach the following functions to events:

- System Functions
- Business Functions

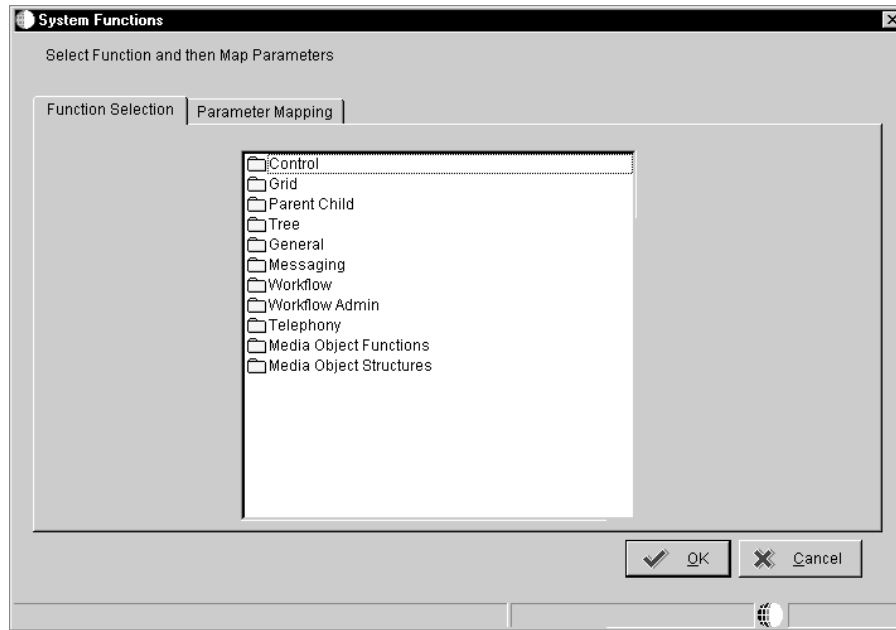
Attaching a System Function to an Event

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:

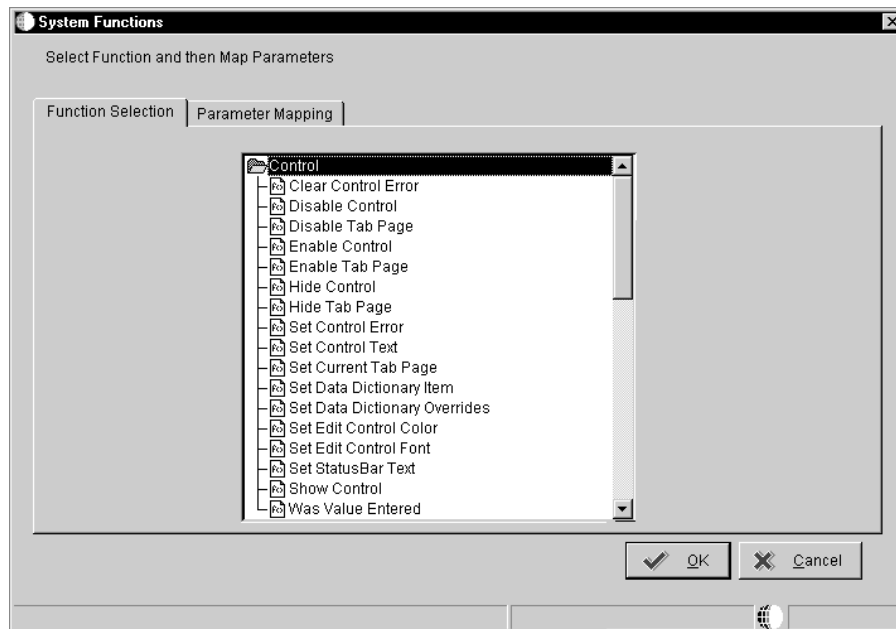
- Hide or display a control
- Insert dates
- Display media objects

To attach a System Function

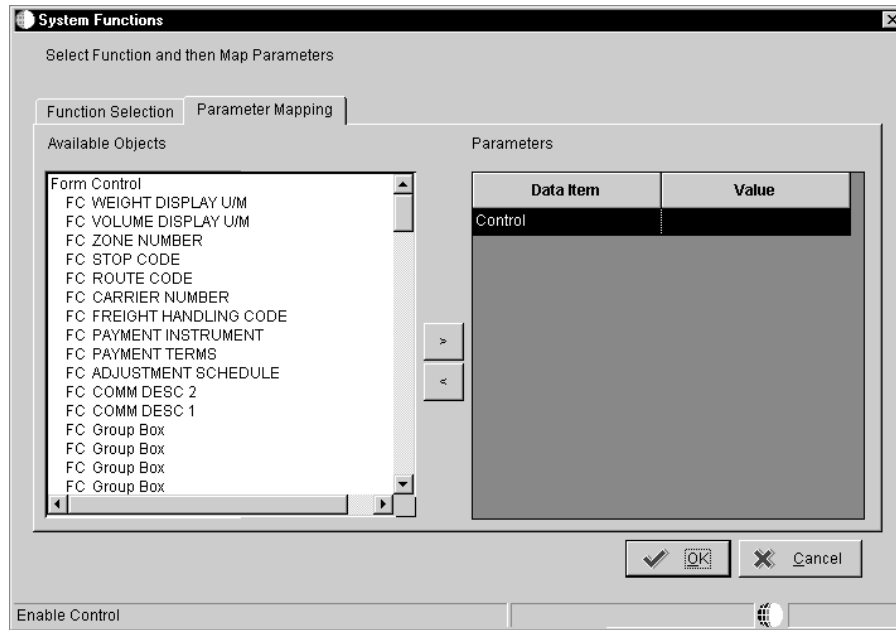
1. On Event Rules Design, choose an event.
2. Click the *f(S)* button.
3. Choose a category in the System Functions box.



4. Choose the system function 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 updatable 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.

See *Telephony System Functions* in the 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:

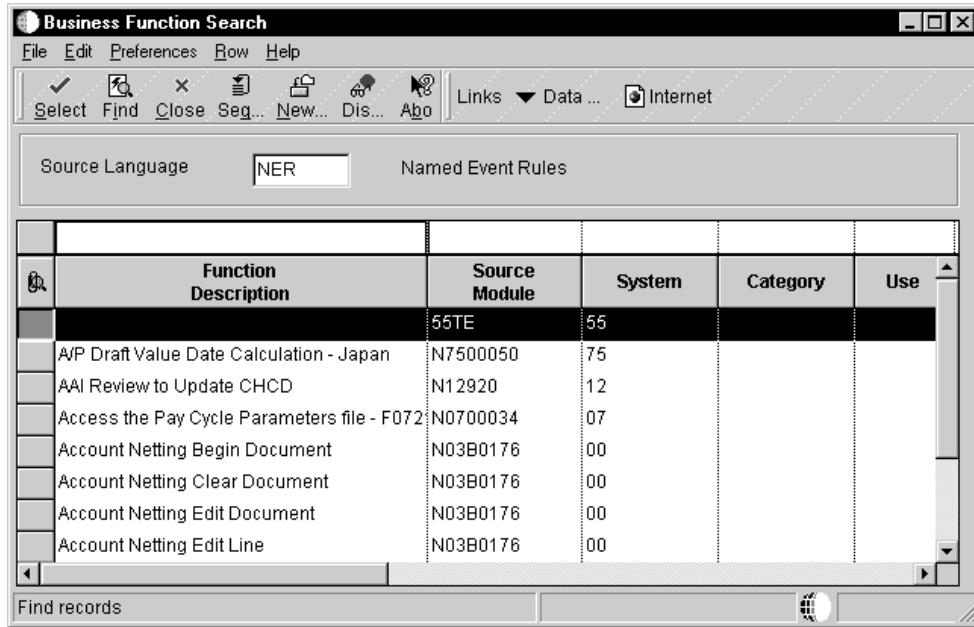
- Manually generated C code (source language C)
- OneWorld generated using Business Function Event Rule Design (source language NER)

Business functions are:

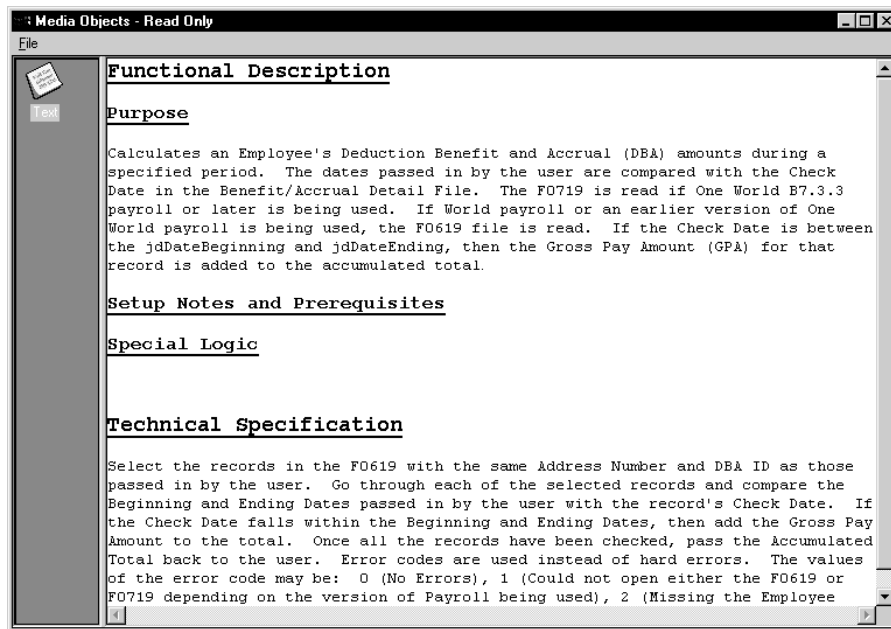
- Commonly used for referential integrity, such as deleting secondary records when a master record is deleted, and for editing routines
- Typically used for large and complex calculations that can otherwise overload the engine.

► **To attach a business function**

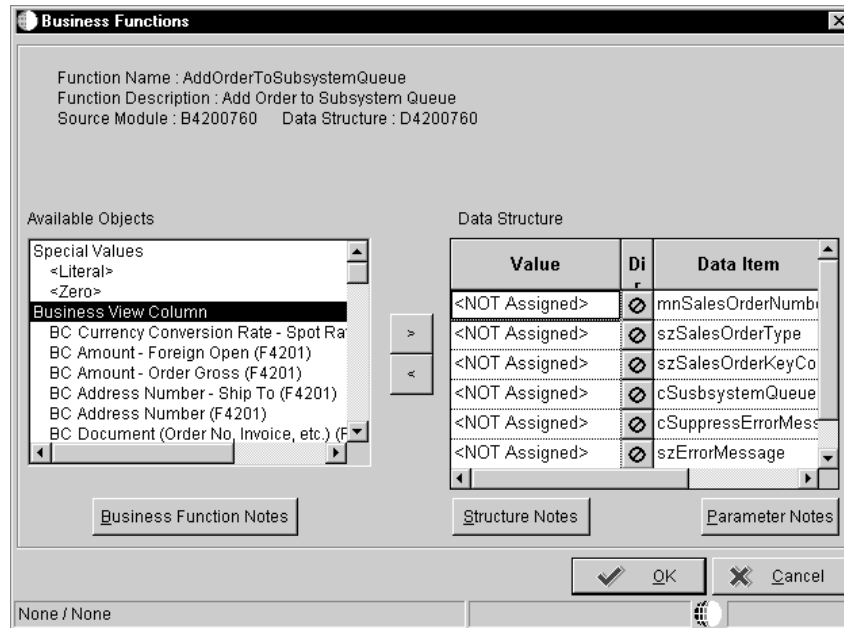
1. On Event Rules Design, choose an event.
2. Click the $f(B)$ button.



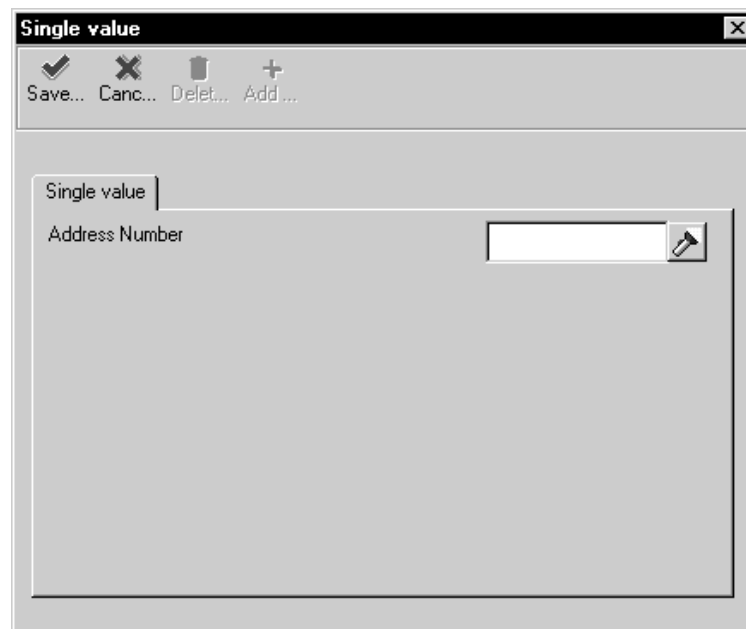
You can view a description for a business function by choosing Attachments from the Row menu.



3. Choose a business function and click OK.
4. In the Available Objects list, choose objects to pass to the business function.



- 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.

- 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.

8. 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.

9. 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.

10. You can click one of the following buttons to add notes.
 - Business Function Notes
 - Structure Notes
 - Parameter Notes
11. 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

Before You Begin

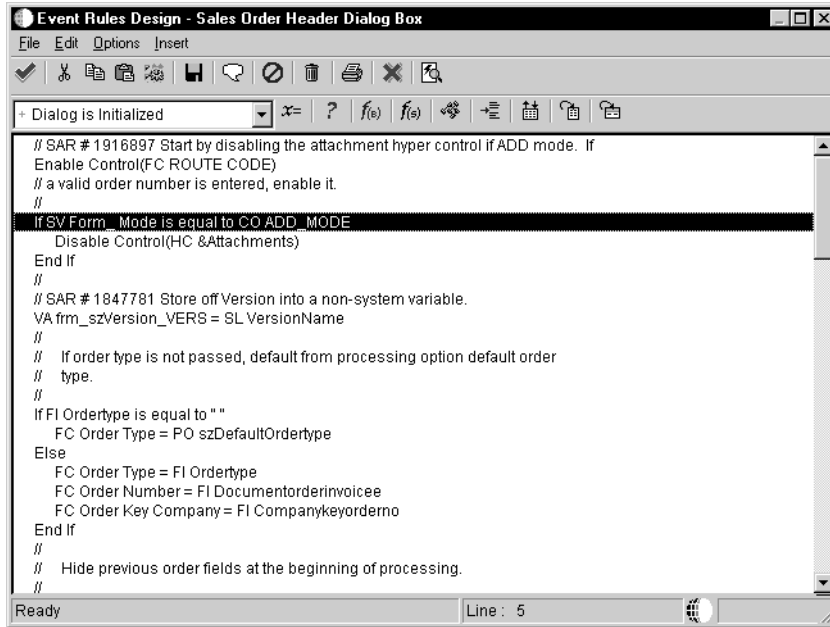
- Before you create a form interconnection, you must define the data structure of the receiving form to include any field for which values are being passed.

Creating a Modal Form Interconnection

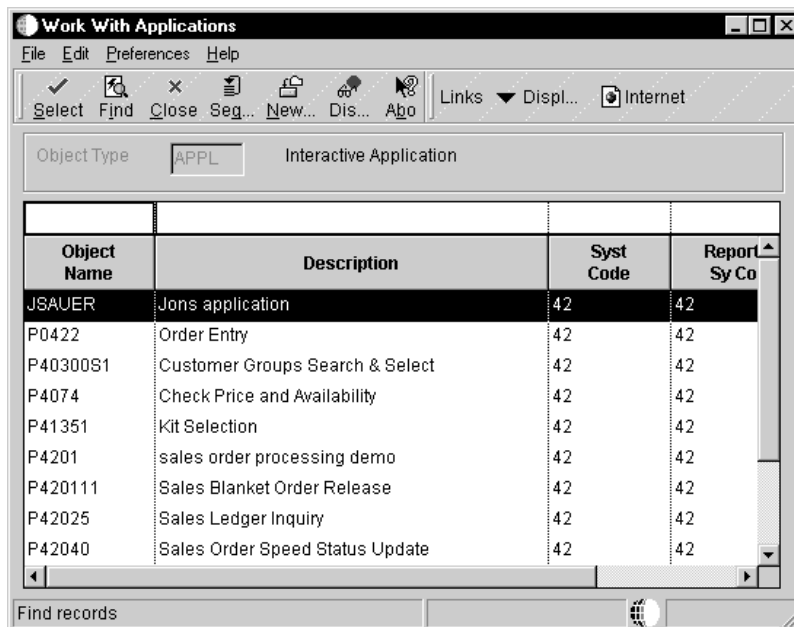
Most form interconnections are modal. This means that after a form interconnects to another form, the user must close the second form before interacting with the first form again.

To create a modal form interconnection

1. On Event Rules Design, choose an event.

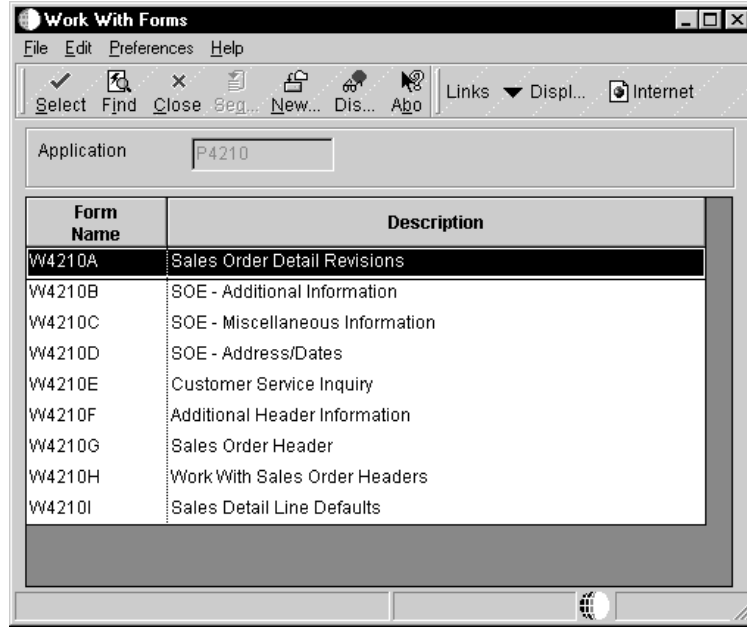


2. Click the Interconnect button.

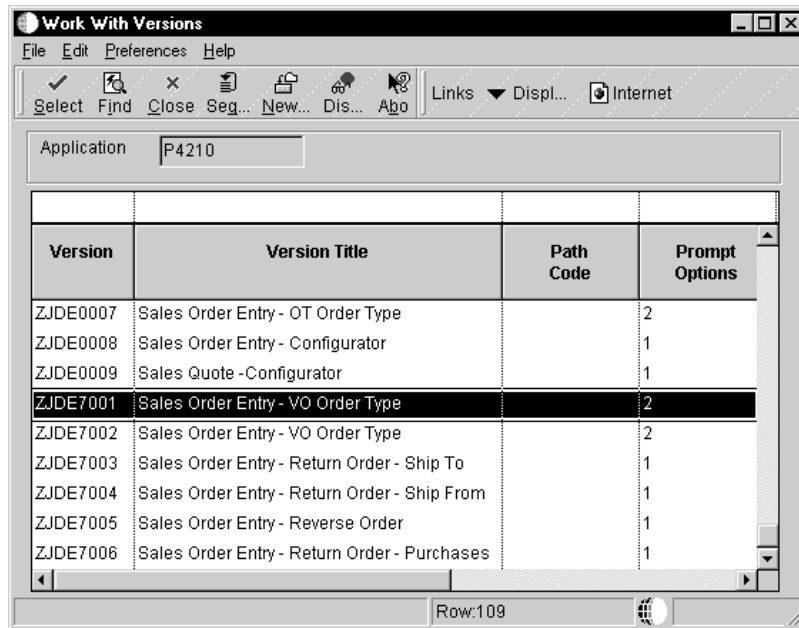


3. On Work with Applications choose the application to which you are connecting.

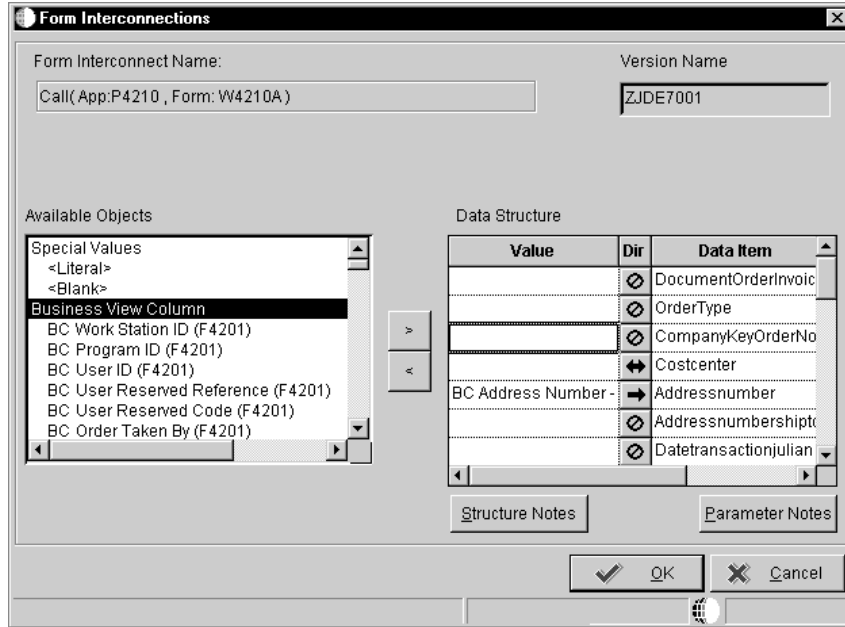
Work with Forms displays forms for the selected application.



4. Choose the appropriate form to which you want to connect (the target).
5. Choose the appropriate version of the form to which you want to connect.



The Data Item column displays data items in the data structure of the target form. The keys in the primary unique index for the primary table of the business view are automatically set up as the data structure.



6. In the Available Objects column, select the object that you want to pass. Use the > button to move it to the Data Structure-Value Column.
7. 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

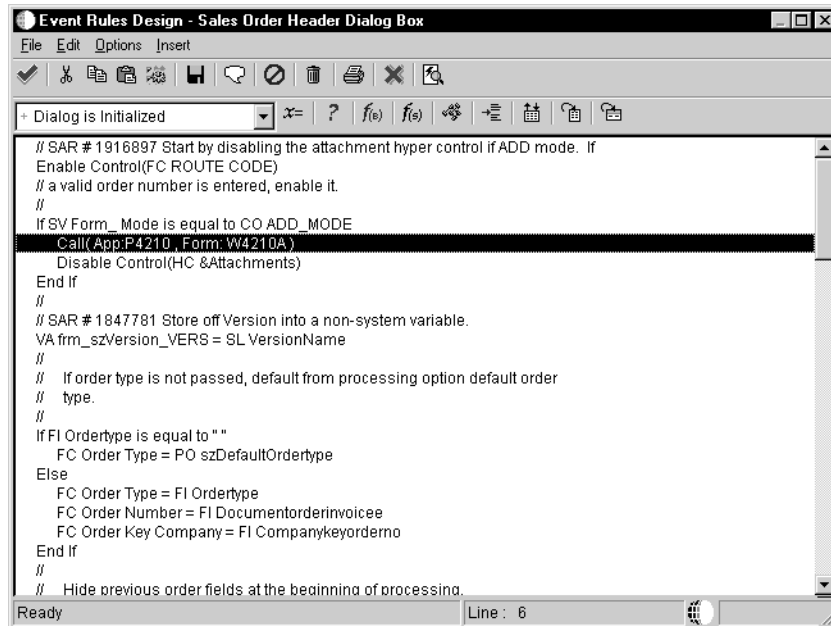
8. Turn the Include in Transaction option to include this interconnection for transaction processing.

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

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.

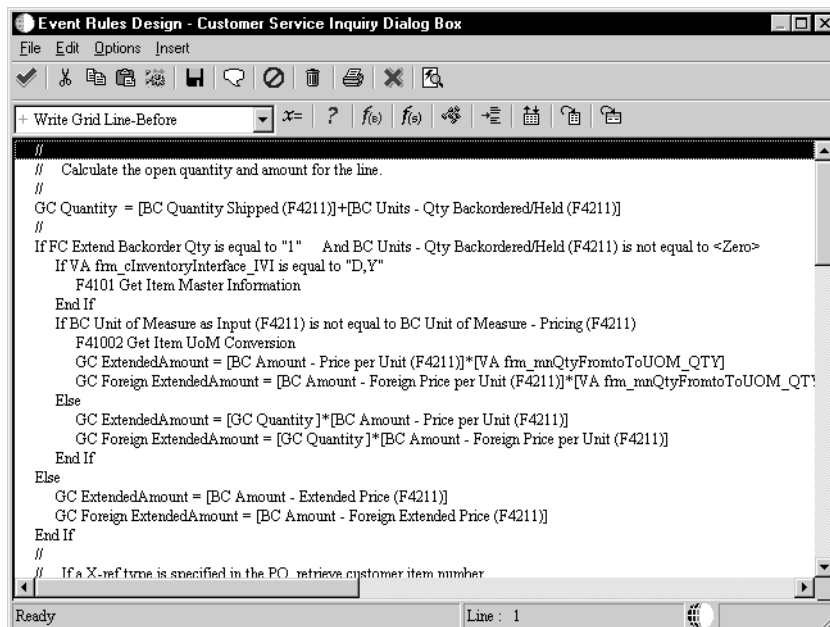
If a Find/Browse form is closed, it goes through its list of modeless interconnect records and closes them before destroying itself.

Event rules following a modeless form interconnection are executed immediately instead of waiting for the called form to return.

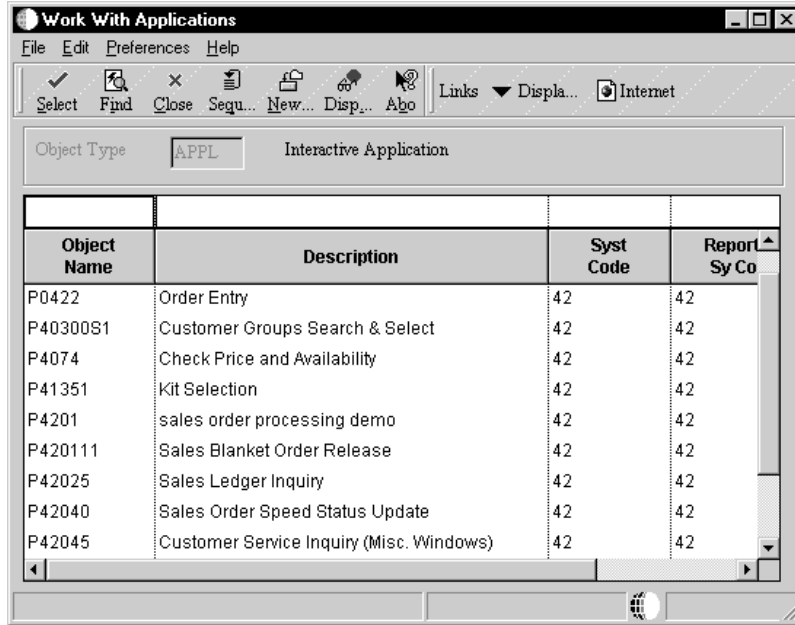
Run applications with modeless processing from the menu. If you run an application from Object Librarian, modeless processing does not work.

► **To create a modeless form interconnection**

1. On Event Rules Design for the Browse form you wish to use (the source), choose an event.

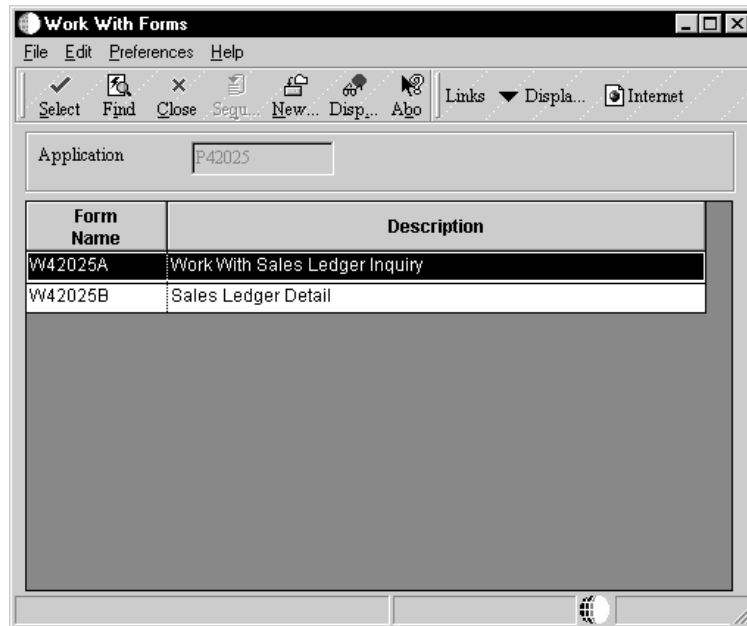


2. Click the Form Interconnection button.

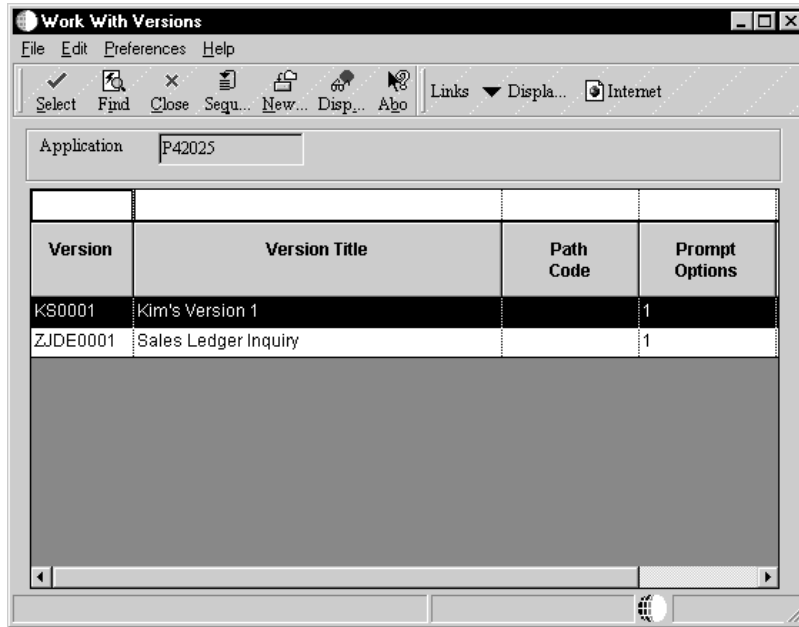


3. On Work with Applications, choose the application to which you are connecting.

Work with Forms displays forms for the selected application.

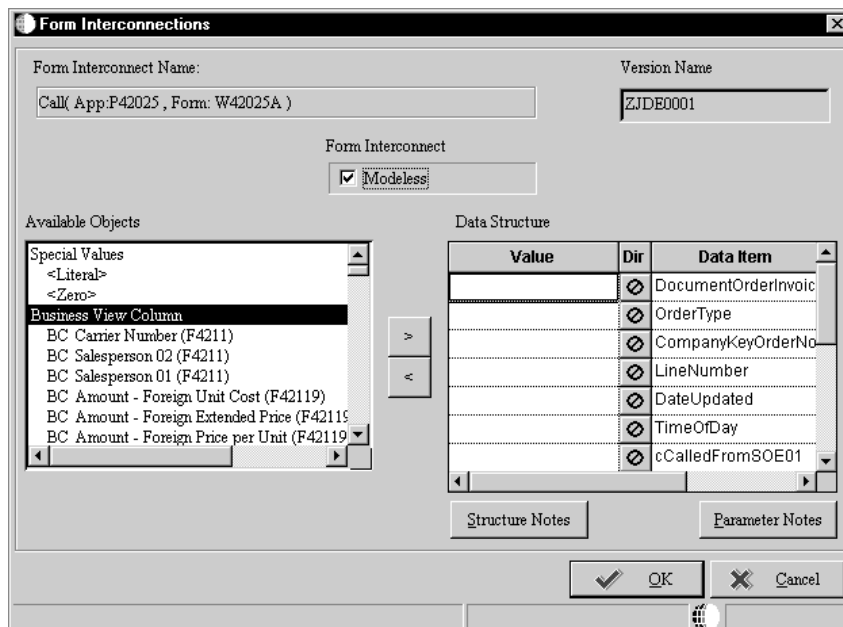


4. Choose the appropriate Fix/Inspect form to which you want to connect (the target). The Form Interconnect - Values to Pass window displays the data structure for the target form.



5. Choose the appropriate versions of the Fix/Inspect form to which you want to connect.

The Data Item column displays data items in the data structure of the target form. The keys in the primary unique index, for the primary table of the business view, are automatically set up as the data structure.



6. Turn on the Modeless option.
7. 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 \emptyset and click OK to save the Form Interconnection and exit.

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

Data flows from the source to the target

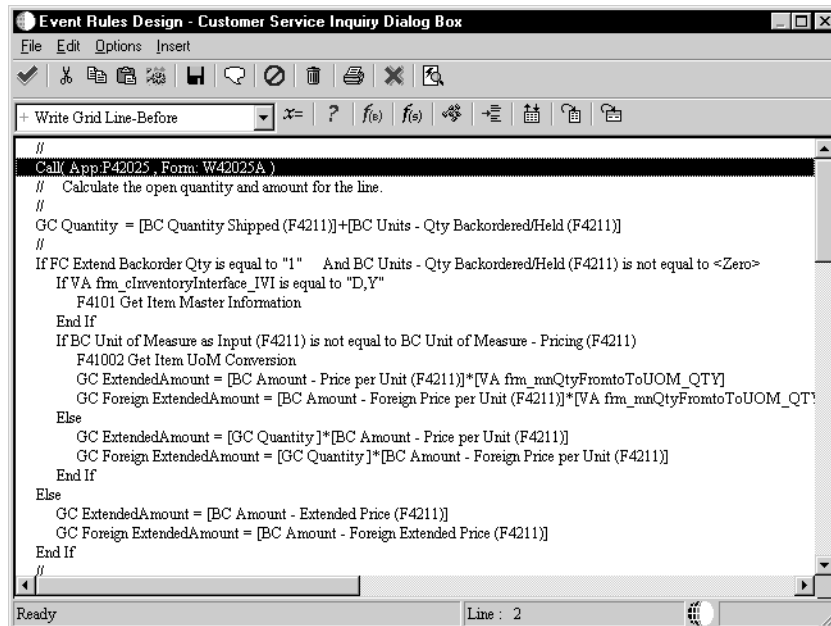
Data flows from the target to the source

Data flows from the source to the target; and upon exiting the target, data flows back to the source

No data flow

9. Click one of the following buttons to add notes:
 - Structure Notes
 - Parameter Notes
10. After 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

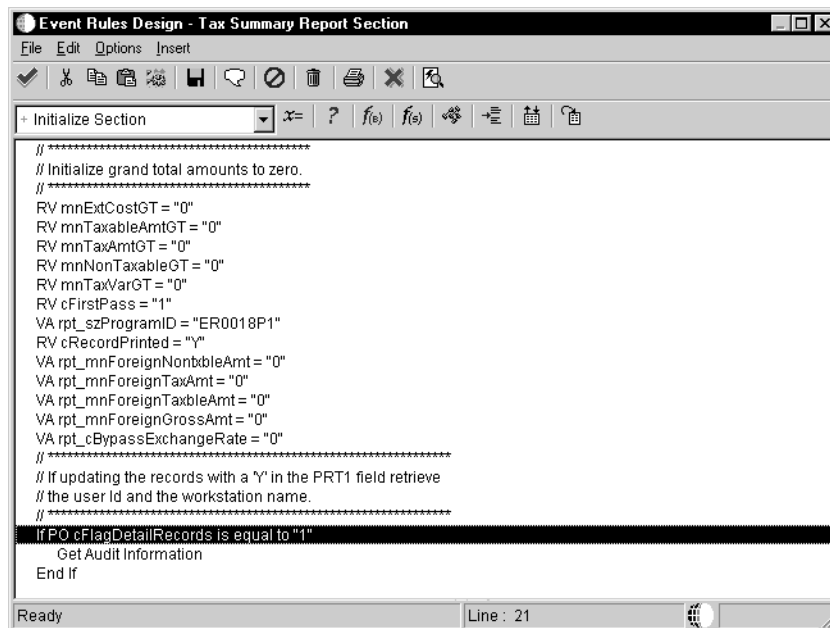
Use report interconnections to automatically execute a report. The current event rules might continue processing or wait for the completion of the report, based upon whether asynchronous processing is enabled. If you use a synchronous report interconnect, your initiating process starts a second process 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.

Before You Begin

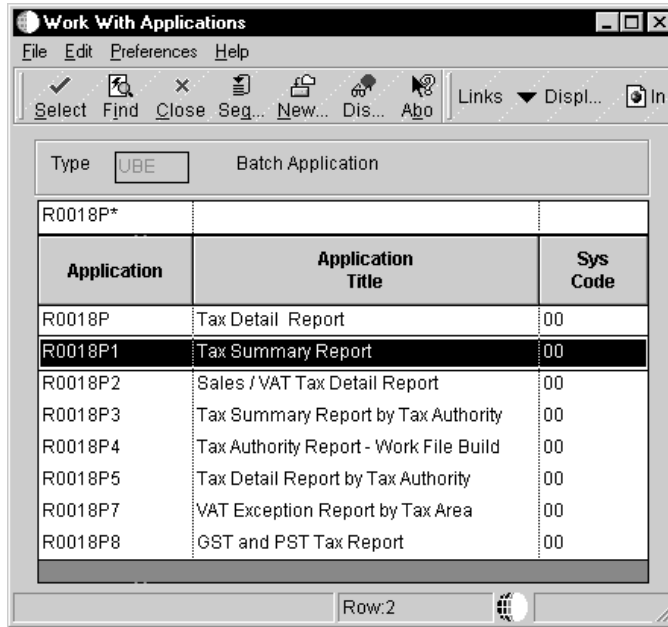
- Before you create a report interconnection, you must define the data structure of the receiving form to include any field for which values are being passed.

► To create a report interconnection

1. On Event Rules Design, choose an event.

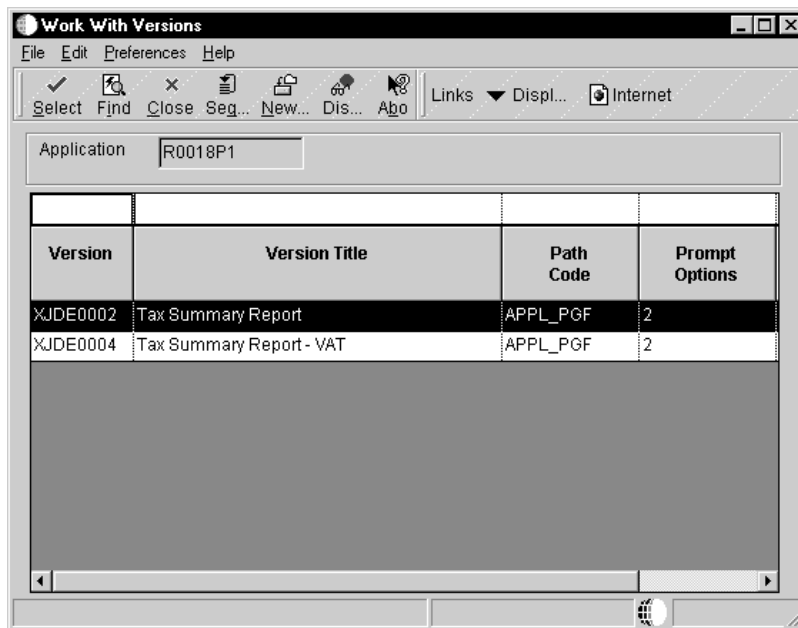


2. Click the Report Interconnection button.

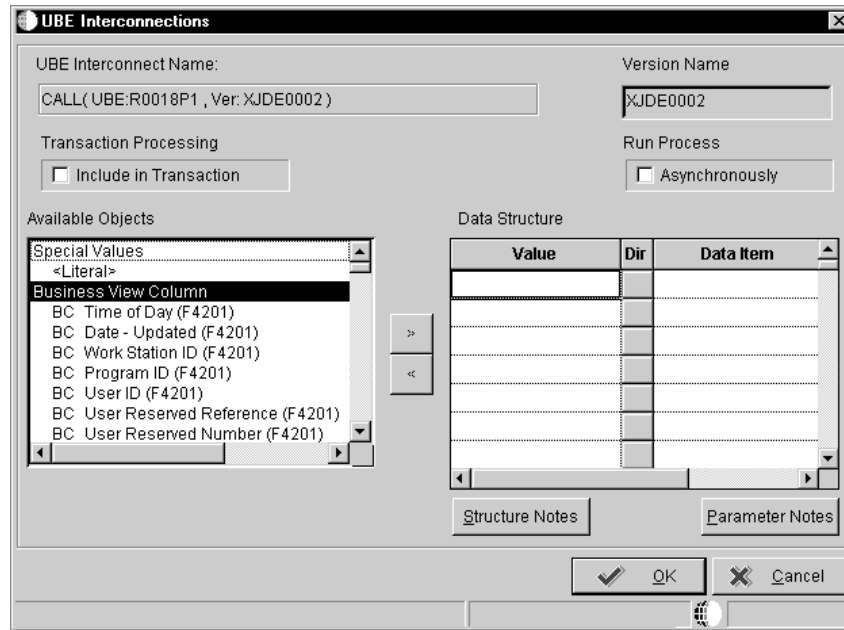


3. On Work with Applications, choose the report to which you are connecting.

Work with Versions displays versions for the selected report.



4. Choose the appropriate version of the report to which you want to connect.



5. In the Available Objects column, choose the object that you want to pass. Use the > button to move it to the Data Structure-Value Column.
6. Indicate the direction of data flow between Value and Data Items.

If you do not want data to pass between reports set all Direction values to ∅ and click OK to save the 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

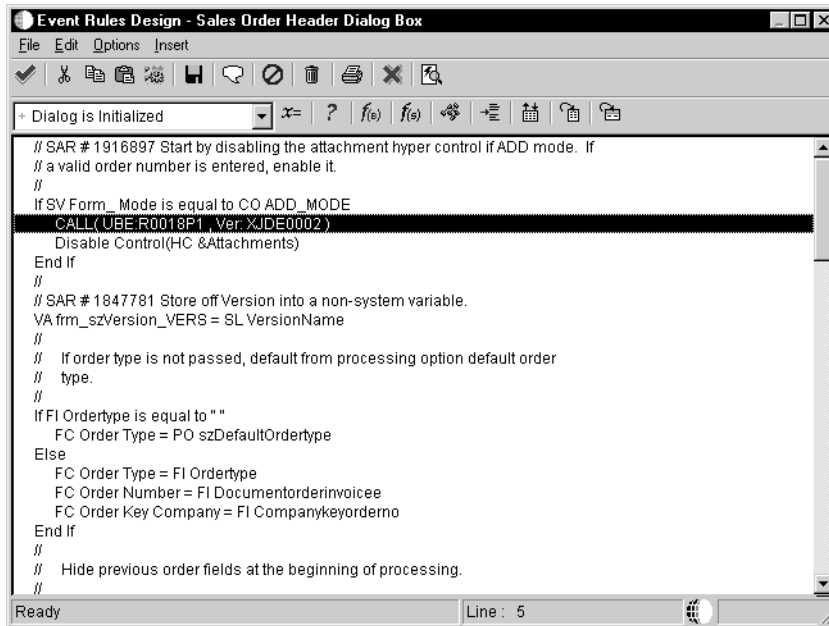
7. To run the report as a separate process, click the following option:
 - Asynchronously

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>).



```
// SAR # 1916897 Start by disabling the attachment hyper control if ADD mode. If
// a valid order number is entered, enable it.
//
// If SV Form_Mode is equal to CO ADD_MODE
// CALL(UBE R0018P1 ,Ver: XJDE0002)
// Disable Control(HC &Attachments)
// End If
//
//
// SAR # 1847781 Store off Version into a non-system variable.
// VA frm_szVersion_VERS = SL VersionName
//
//
// If order type is not passed, default from processing option default order
// type.
//
//
// If FI Ordertype is equal to ""
// FC Order Type = PO szDefaultOrdertype
// Else
// FC Order Type = FI Ordertype
// FC Order Number = FI Documentorderinvoicee
// FC Order Key Company = FI Companykeyorderno
// End If
//
//
// Hide previous order fields at the beginning of processing.
//
//
```

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:

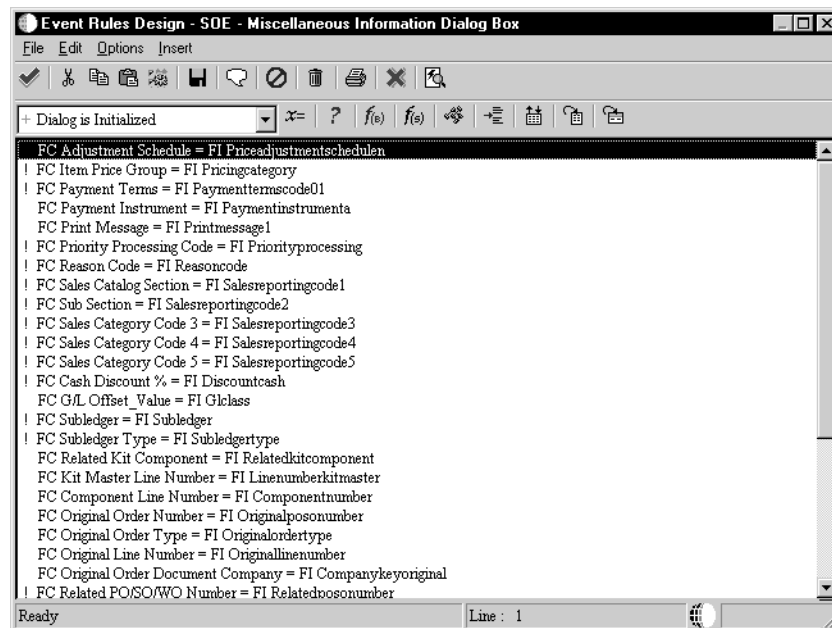
- Assign a fixed value
- Create and assign a mathematical expression

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.

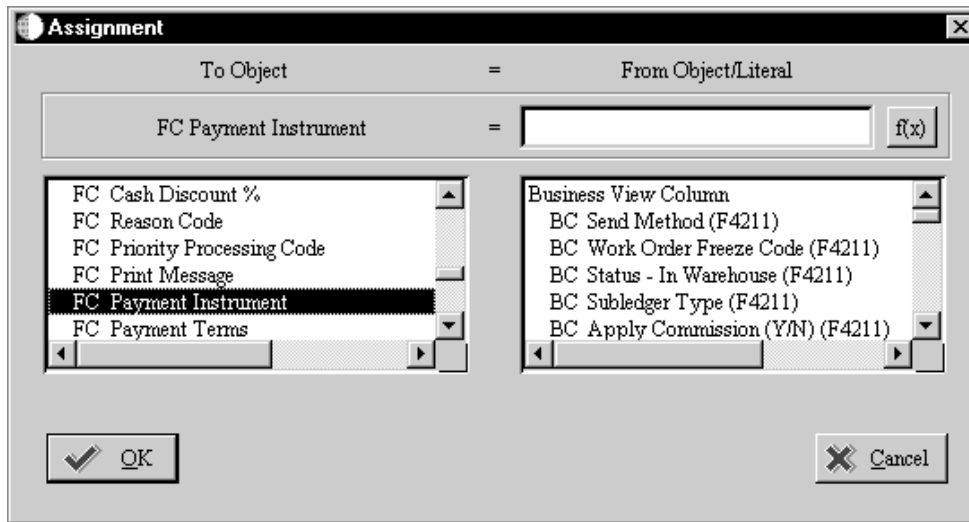
The available objects that appear are based on the data type.

► To assign a value

1. On Event Rules Design, choose an event.



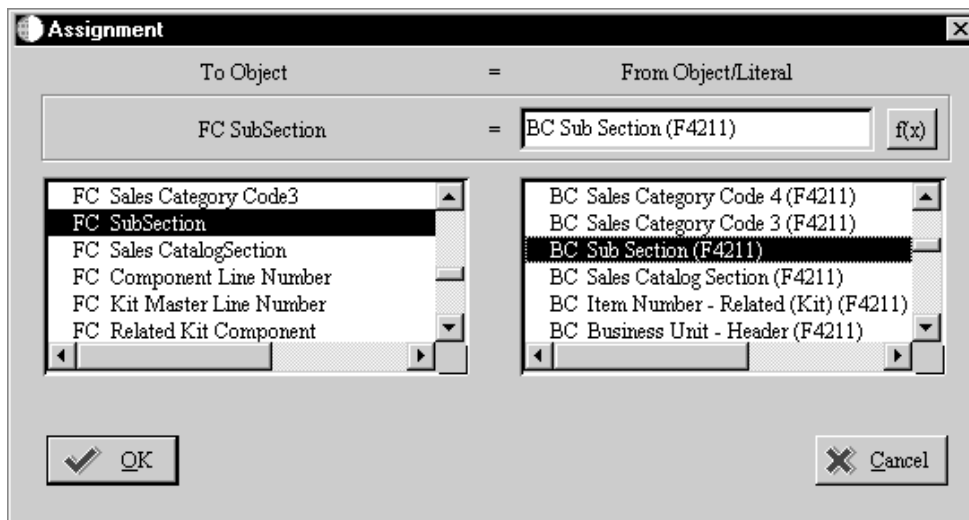
2. Click the Assign button.



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.

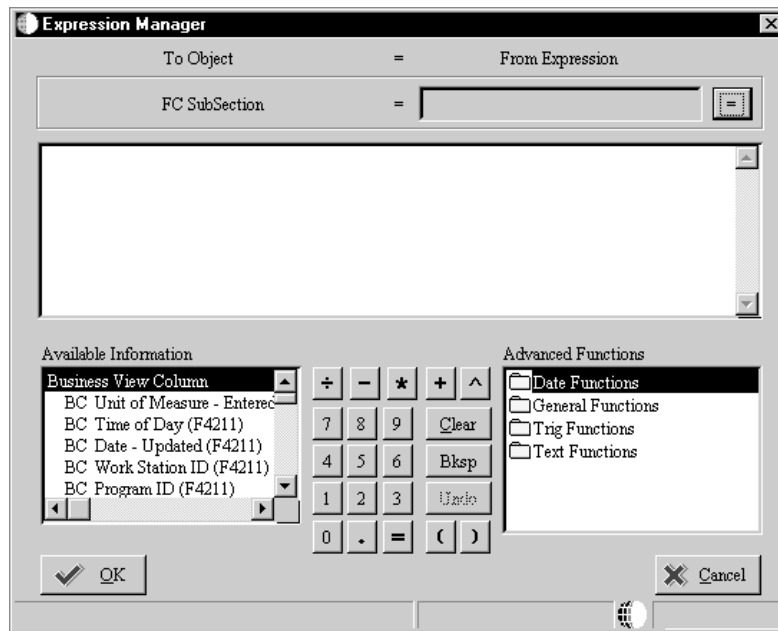


5. Click OK.

The Event Rules Design window displays the assignment in text form.

► **To create an expression for an assignment**

1. From the Assignment form, 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:

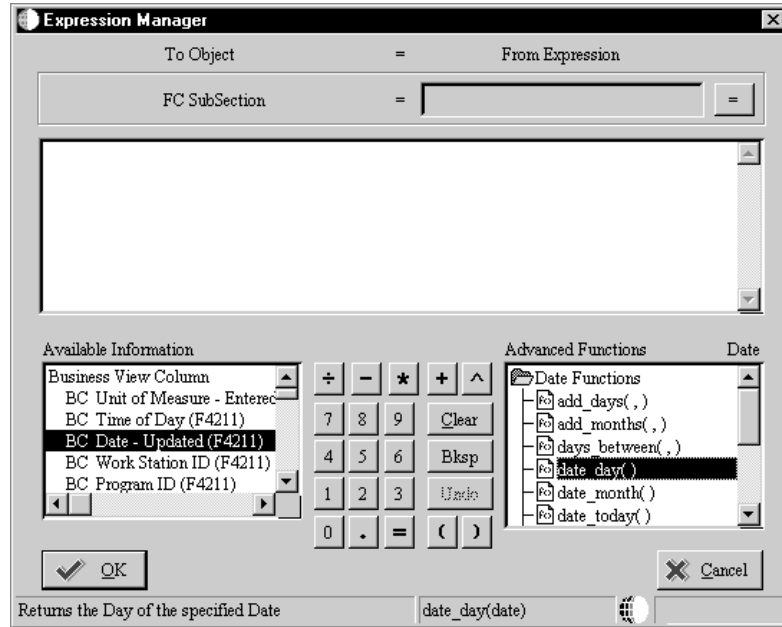
- | | |
|--------------|--|
| Clear | Clears all data from the Assignment Display |
| Bksp | Deletes the previous expression or character |
| Undo | Reverses your previous editing action (other than inserting objects or functions from the lists) |

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 lower("AbCdEfG") = "abcdefg"
lpad	Inserts the character into the front (left side) of the string until the string is length long lpad("qwerty", ' ', 12) = "..... qwerty"
ltrim	Removes leading occurrences of the character from the string ltrim("aaaaaabcdef", 'a') = "bcdef"
rpadd	Inserts the character into the back (right side) of the string until the string is length long rpadd("qwerty", ' ', 10) = "qwerty..."
rtrim	Removes trailing occurrences of the character from the string rtrim("abcdef ", ' ') = "abcdef"
substring	Extracts a portion of string beginning at position for length number of characters substring("SUNMONTUEWEDTHUFRIS AT", 9, 3) = "WED" 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 upper("qweRTY") = "QWERTY"

Table I/O

Use the Table I/O button in Event Rules Design to create instructions that perform table input and output (I/O), without having to manually code a business function in C code. Table I/O allows you to access a table through event rules. For example, you can use table I/O to display information in a table that your application does not use. 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

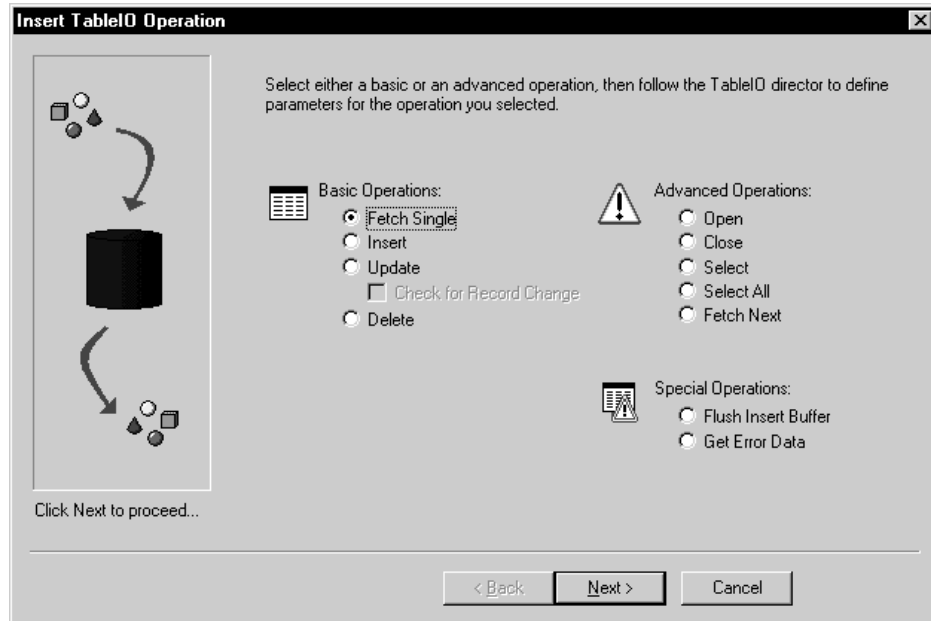
You can use the following operators for mapping specific table I/O operations:

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

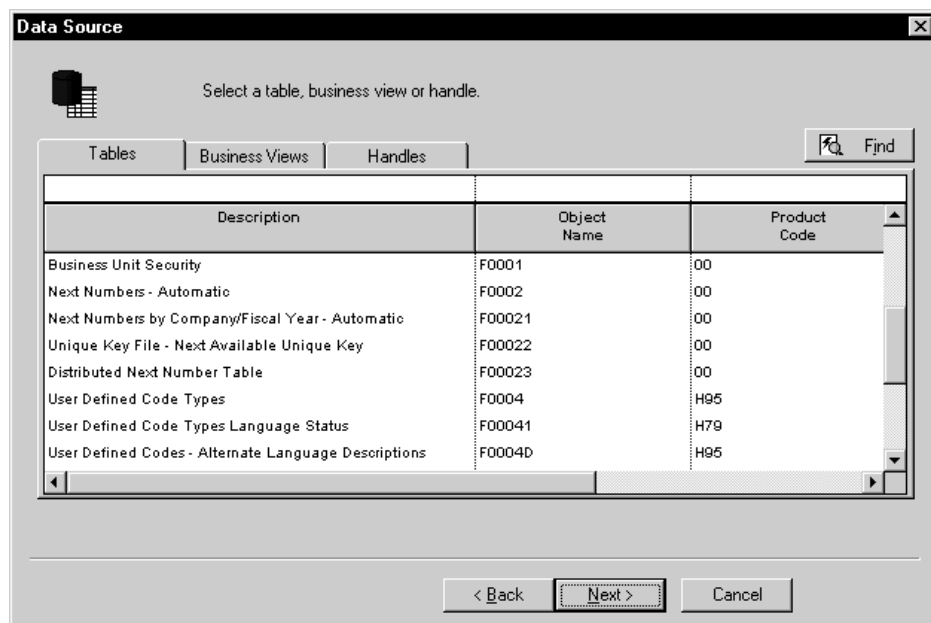
Creating a Table I/O Event Rule

► **To create a table I/O event rule**

1. On Event Rules Design, click the Table I/O button.



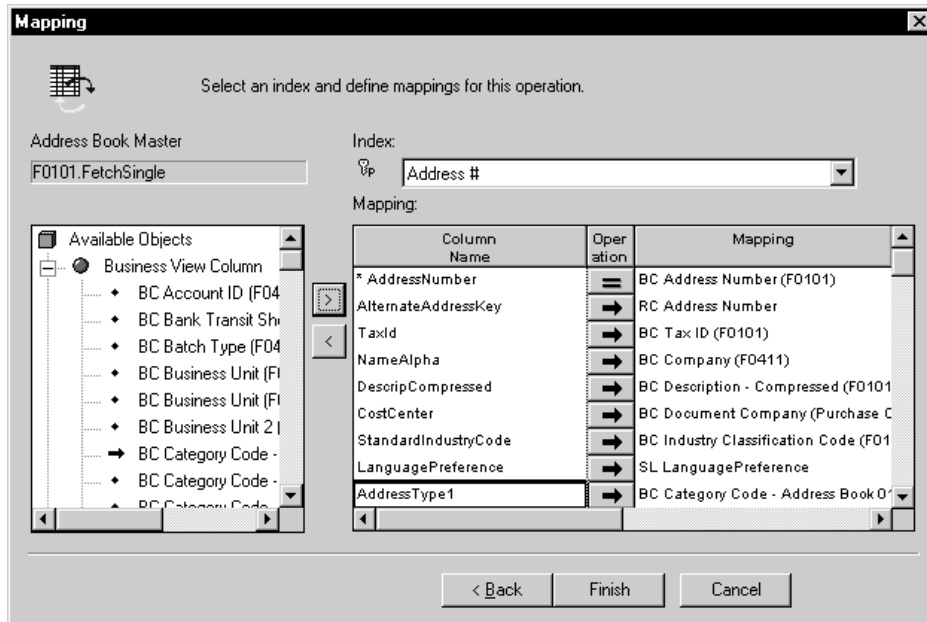
2. Choose the operation you wish to use, and then click Next.



3. On Data Source, choose the table or business view or handle for which you want to perform I/O, and then click Next.

The Mapping form displays available objects that you can map to selected table columns. The available objects are used to build a WHERE clause on SELECT statements. 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.



4. Choose the column you want to use and then double-click on the available objects you wish to map to that column.
5. Click the Operator button until you have the operation you want.

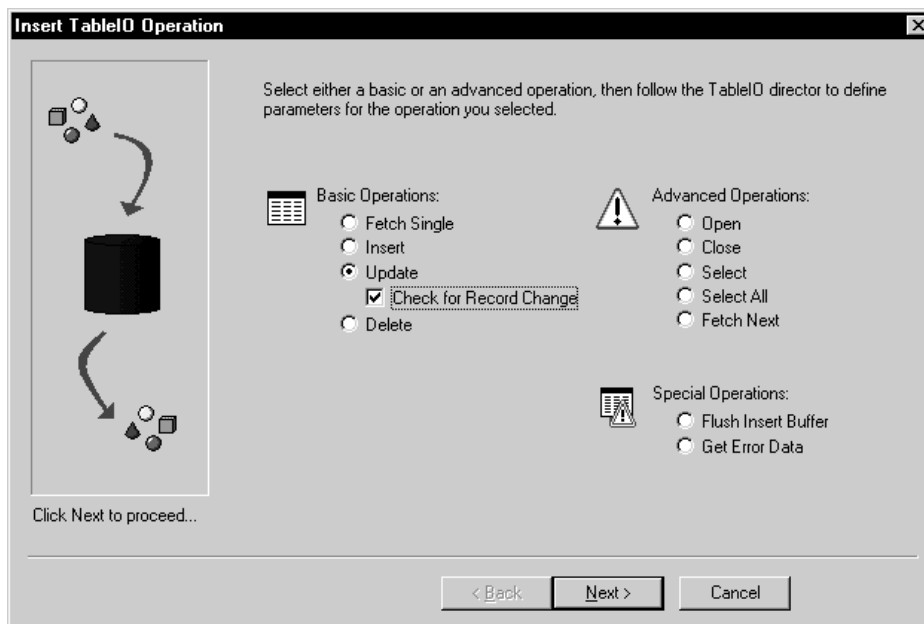
The following selection operations are available. The default is equal.

- Equal
- Not Equal
- Less Than
- Less Than or Equal To
- Greater Than
- Greater Than or Equal
- Like

6. Click Finish to save the operation and return to Event Rules Design.

Record Change

If you choose the Update option, you can also enable Check for Record Change. The *Check for Record Change* option reports if the record has been changed between the time that it was fetched and the time that it is updated.



Understanding Buffered Inserts

You can use buffered inserts to improve performance when you insert many (hundreds or thousands) of records into a single database table, and you do not need immediate user feedback if there is an insertion failure. You can use buffered inserts with table conversion, table I/O, batch processes, and business functions. They are not available for interactive applications. Buffered inserts are only available with Oracle (V8 and above), DB2/400, and SQL Server. Buffered inserts are not available with Access, post-insert triggers, or multiple-table views. The JDEbase database middleware delivers records to the database management system one bufferload at a time.

When you request buffering, the database records are inserted individually and the buffer is automatically flushed when it fills; that is, the JDEbase database middleware delivers the buffer to the database management system. The buffer can also be explicitly flushed. For example, the buffer flushes automatically when a transaction is committed or when a table or view is closed. The business function, table conversion engine, or table I/O can explicitly request that the buffer be flushed as well.

Buffered Insert Error Messaging

Use caution when you decide whether to use buffered inserts. Remember that there is no immediate feedback if an insertion fails. Instead, the error shows in the JDE log. Consequently, buffered inserts are used primarily with batch applications.

Unless using the Table Conversion application, you must request more detailed information from the middleware to get detailed error messages. In Table

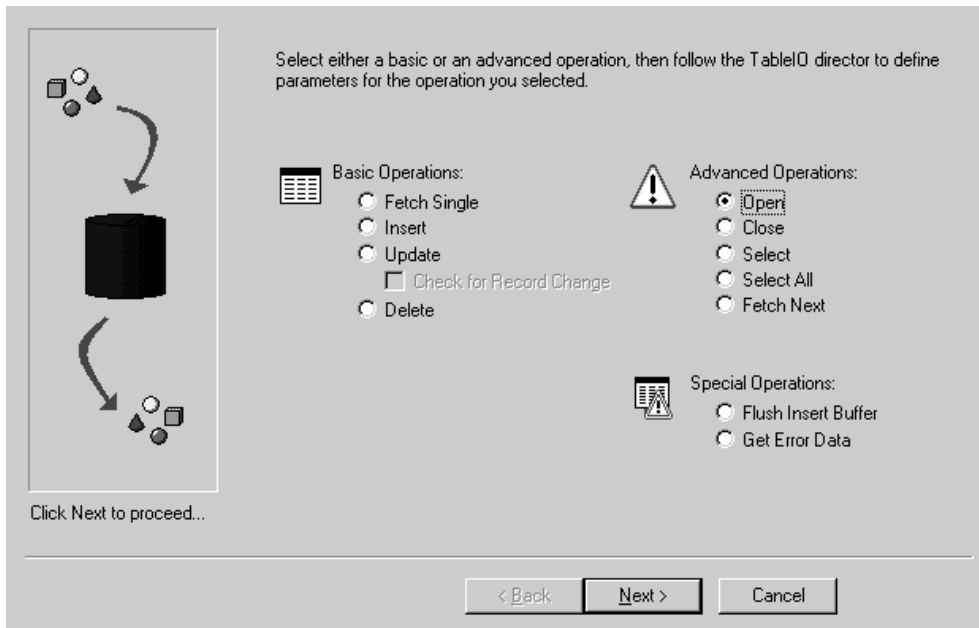
Conversion, this is performed automatically by the table conversion engine. Turn on tracing to get more detailed error messages. Otherwise, you get an error message that the insert failed. Clear the output tables so that you do not get duplicate error logging.

The typical process you follow to use for buffered inserts and retrieving error messages is as follows:

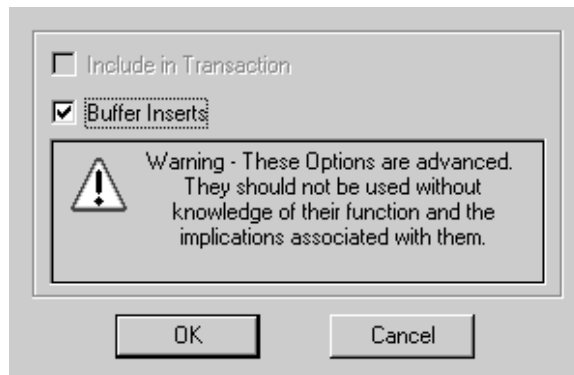
Following are three examples of messaging for the same buffer flush. The first result had error messaging and tracing turned off. The second had error messaging turned on. The third had error messaging and tracing turned on.

Using Buffered Inserts in Table I/O

When you use buffered inserts in table I/O, you must explicitly open the table with an Open.



Choose the table you wish to use and click Advanced Options.



Special Operations

After you have buffered inserts set up, you can use Special Operations to flush the buffers or get error messages. In Event Rules, at the point that you want to perform one of these operations, click either of the following options:

- Flush Insert Buffer

To maintain data integrity, you should flush the insert buffer before you perform any operations other than an insert. If you fail to do so, the results of recent inserts may not be reflected in other operations and the operations may not work properly.

When you use the flush insert buffer option for a specific table you must flush the buffer before you close the table so that you still have access to error information. You can use the Get Error Data option for each insert to ensure that you get this information.

- Get Error Data

The Get Error Data option retrieves errors for records that did not insert properly. You should be aware that depending on when your buffers get flushed, or when you begin another insert, you may overwrite the error information for a specific insert. If error information is critical you should retrieve the information before the next insert begins.

If you need special error handling performed, you should set it up after each table I/O insert and each Flush Insert Buffer option. Always retrieve the error information before you begin your next table operation.

Understanding Handles

In OneWorld, the table I/O term handle refers to a type of file pointer. This file pointer connects the OneWorld application or UBE with the middleware used to communicate with the database manager. Handles are references to an address within the middleware that point to a database table. Unlike regular file pointers, handles allow you some control over when and how they are used. Handles allow you to perform several operations 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. This is particularly helpful whenever there is an upgrade to OneWorld or when you need to convert data from another system into OneWorld.

- You can pass handles into a form, named event rule, or business function so that you do not need to open a table or business view more than once.

Handles cannot be used in transaction processing.

If you pass a handle to a form or a named event rule, the data structure for the form or named event rule must contain a member that is a handle data item. In the form interconnect or business function call, you must assign a handle value from your event rules to the handle data structure member. You can use this handle in the form or named event rule that is receiving the handle just like any other handle.

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.

Using 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
- Open the handle explicitly

After 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. See *Working with Event Rule Variables* for more information about creating variables.

After you create a handle variable, you must explicitly open the handle. Then, after performing the required table I/O, you must explicitly close it.

To use a handle

Use a Class Type of Handle and a Data Type of 7. You should name the alias for the handle the same as its table.

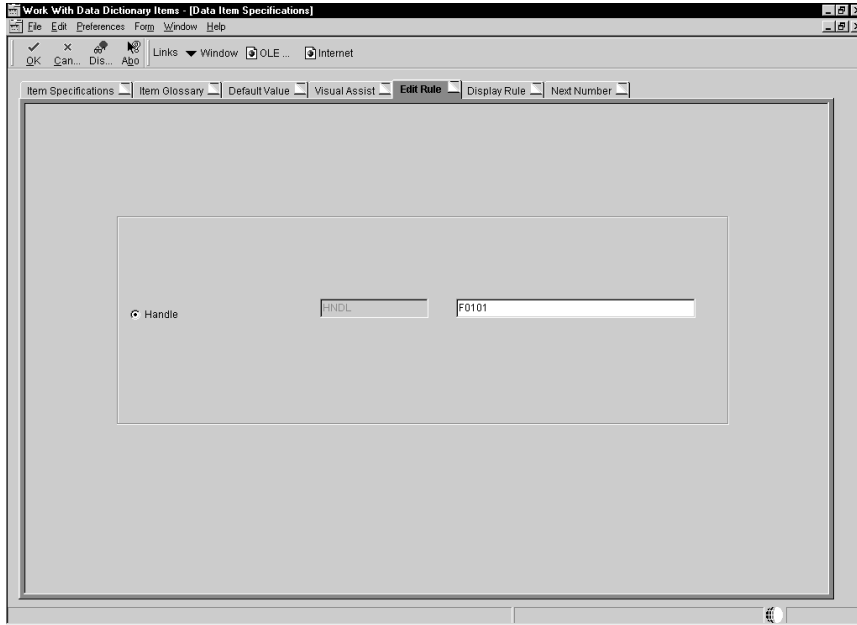
1. To complete the handle data item, click the Edit Rule tab and type in the table or business view name for the handle, then save the data dictionary item.

The data item name can be a maximum of eight characters and should be formatted as follows: HFxxxxxx

HF = designates a table I/O data item

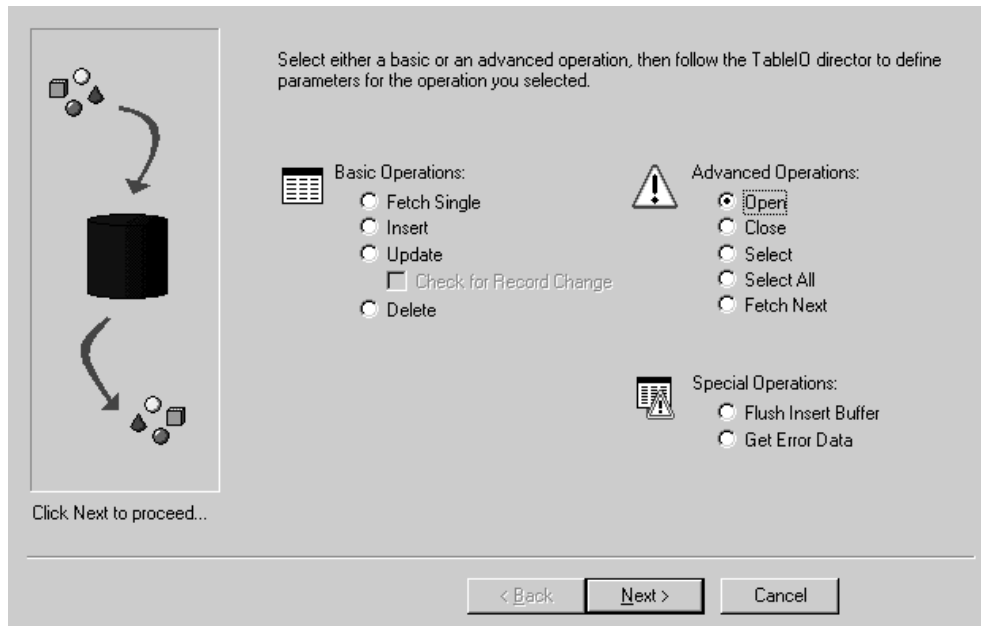
xxxxxx = system code and group type used in the table name

For example, the table I/O data item name for table F4211 would be HF4211.



See *Data Dictionary* for more information about data items. You can also create a handle data item in a data structure.

2. On Event Rules Design, click the Table I/O button.
3. From Advanced Operations, click Open.



4. Click Next.
5. On Data Source, click the Handles tab.
6. Choose the handle you wish to open, and then click Next.

7. Choose a variable that contains the name of the environment in which you want to open the table.

If you want to open the table in the login environment, choose the system value SL LoginEnvironment. System values also exist for the source and target environment in Table Conversion if you use Table I/O in Table Conversion.

8. Click Finish.
9. On Data Source, click the Handles tab.
10. Choose the handle you wish to close and 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:

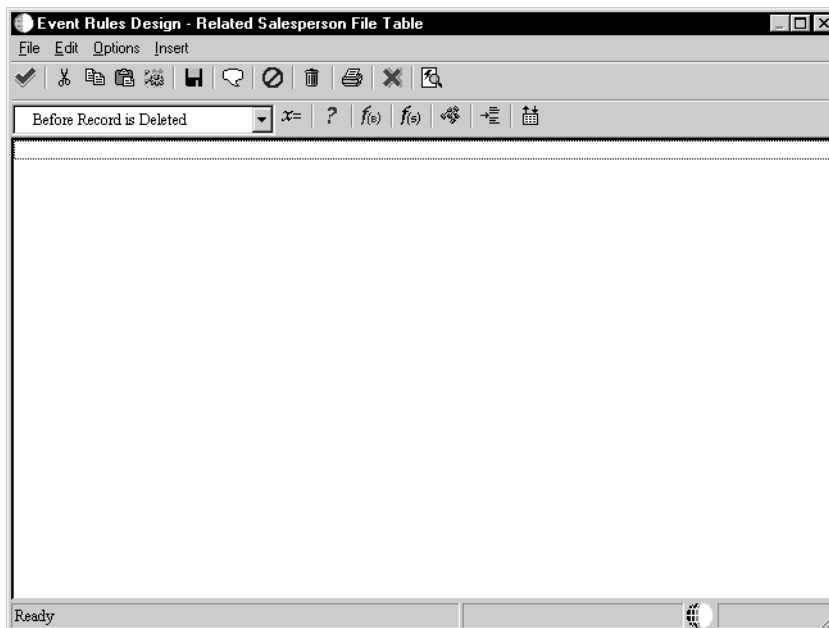
- Create the database trigger in Event Rules Design
- Generate the OneWorld database trigger as C code

▶ To create table event rules

1. On Object Management Workbench, check out the table to which you want to attach event rules.
2. On Table Design, click the Design Tools tab, and then click Start Table Trigger Design Aid.

This starts Event Rules Design, which you can use to attach event rules to any of the events for the table.

3. From the Events list, choose an event.



4. Click one of the following event rule buttons:

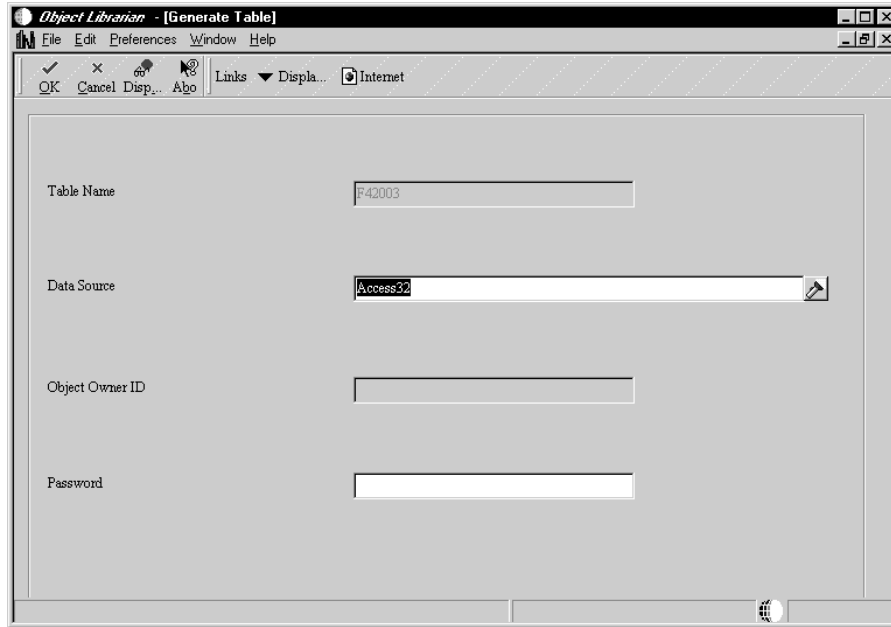
Business Function Attaches an existing business function

System Function Attaches an existing 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.

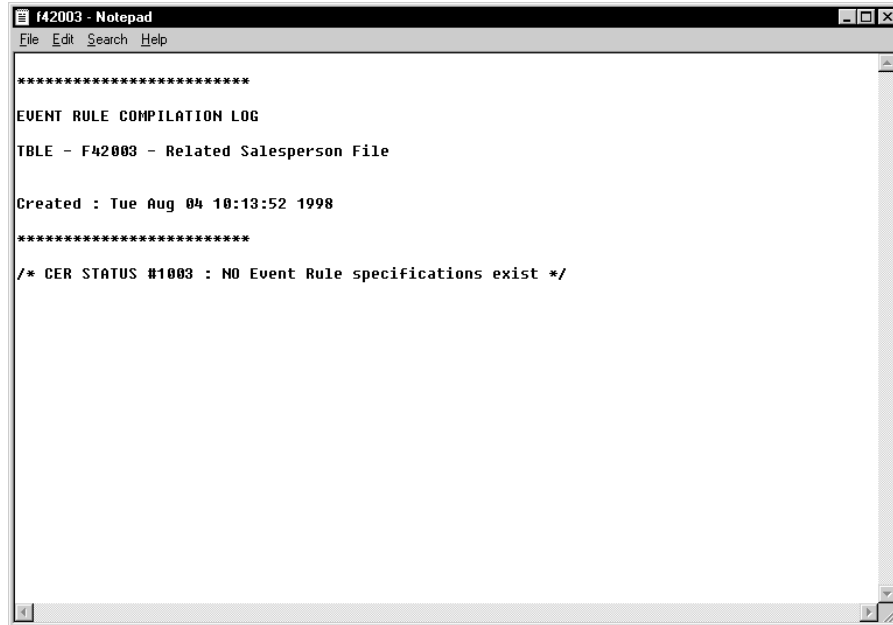
5. On Event Rules Design, click Save to save your event rule specifications, then click Close to return to Table Design.
6. On Table Design, click the Table Operations tab, and then click Generate Table.
7. On Generate Table, complete the following fields, and then click OK:
 - Data Source
 - Password



8. On Table Design, click the Design Tools tab, and then click Build Table Triggers.

The Build Triggers option performs the following steps:

- Converts the 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.
9. To see a log of the build, click Generate Header File, and then open the file the system creates.

A screenshot of a Notepad window titled 'f42003 - Notepad'. The window contains the following text:

```
*****  
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 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.

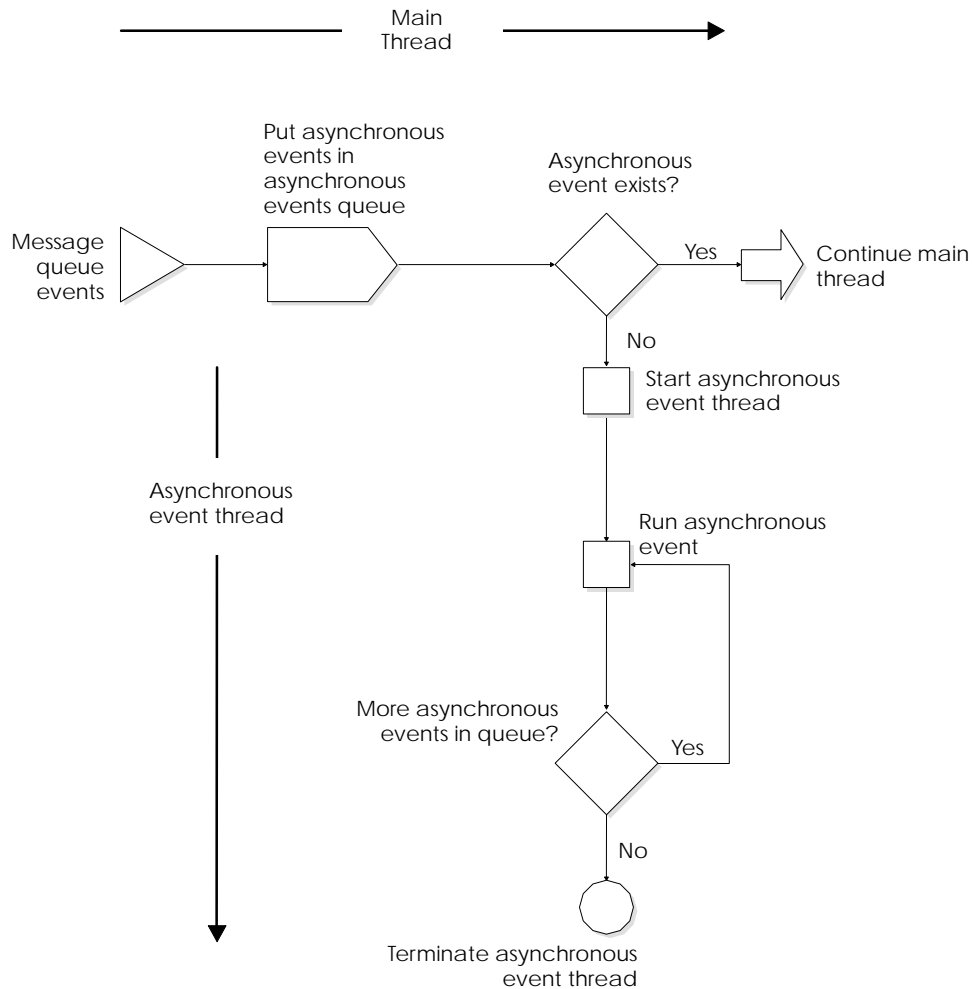
Asynchronous processing does not support form interconnections. Asynchronous processes are used for background processing, and users should not interact with an asynchronous thread directly. For example, if you have an asynchronous process (such as *Row is Exited and Changed - Asynch*) you should not 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 - Async*
- *Row is Exited and Changed - Async*

- *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 - Asynch* events is run on a thread. That means the user will be allowed to continue keying in data while the event is processing.

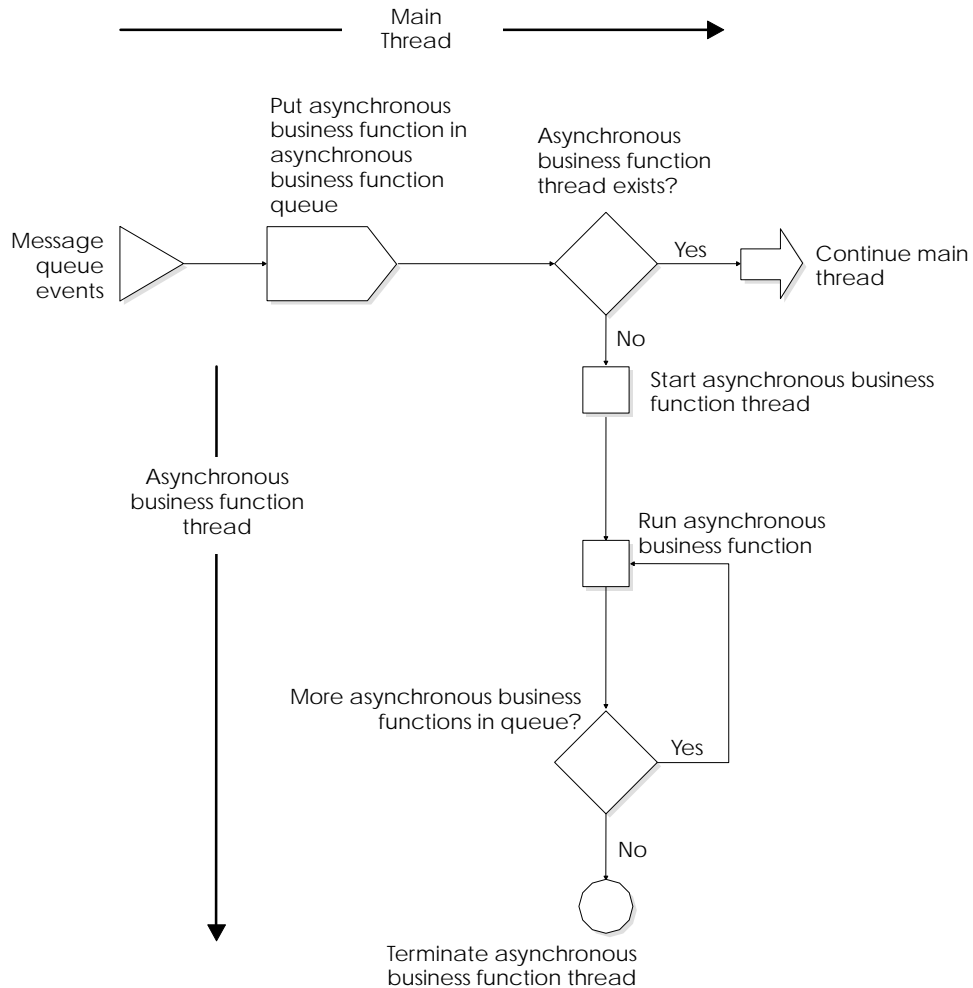
Asynchronous Business Functions

Business functions that are run on OK and Cancel processing are a performance bottleneck. Whenever 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 asynchronous option is disabled for all events except *OK Post Button Clicked*, *Cancel Post Button Clicked*, and *Close Post Button Clicked*. When the option is turned on the business function processes on a thread. When the option is turned on, parameters can be passed in, or not passed at all. 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 are ignored. The asynchronous business function has no connection to the form, so errors cannot be set. The runtime engine can, however, use the return code from `jdeCallObject` to determine if an asynchronous business function fails. A message box appears if a failure occurs. This is important because it allows you to quit processing when a failure occurs. You can check the logs for more information about the failure.

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.

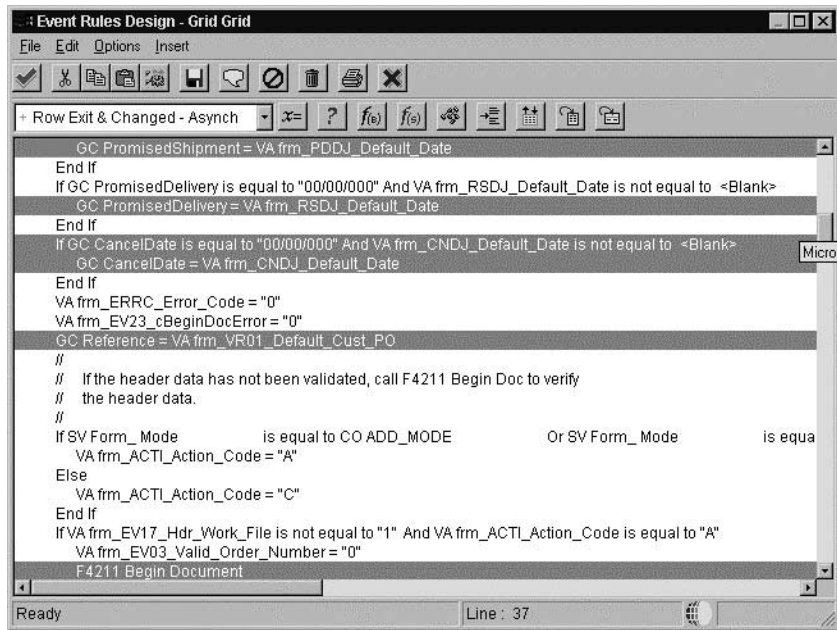
Asynchronous business functions must complete their processing before OneWorld Explorer can be closed. If there are still asynchronous business functions running, a message will be displayed and OneWorld Explorer will not close.

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:

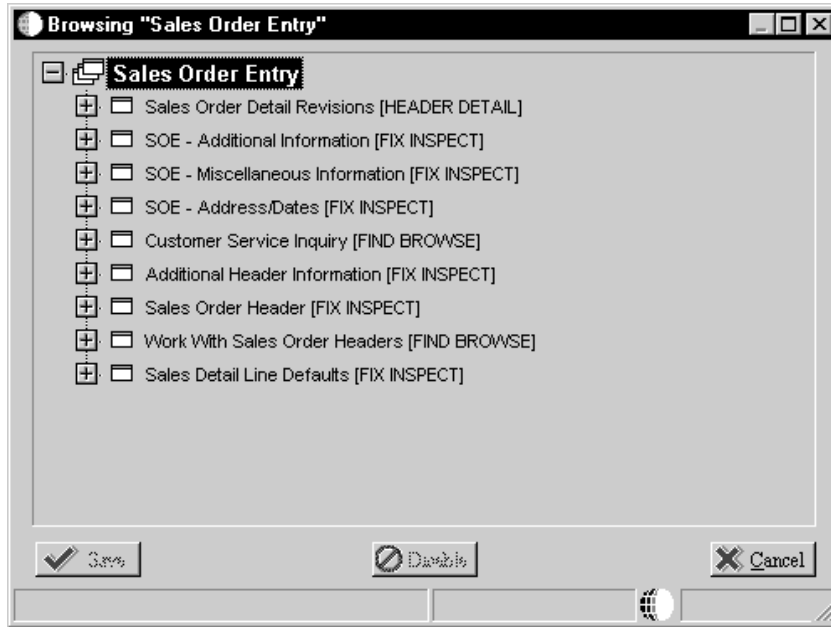
- Working with BrowsER
- Working with BrowsER options

Working with BrowsER

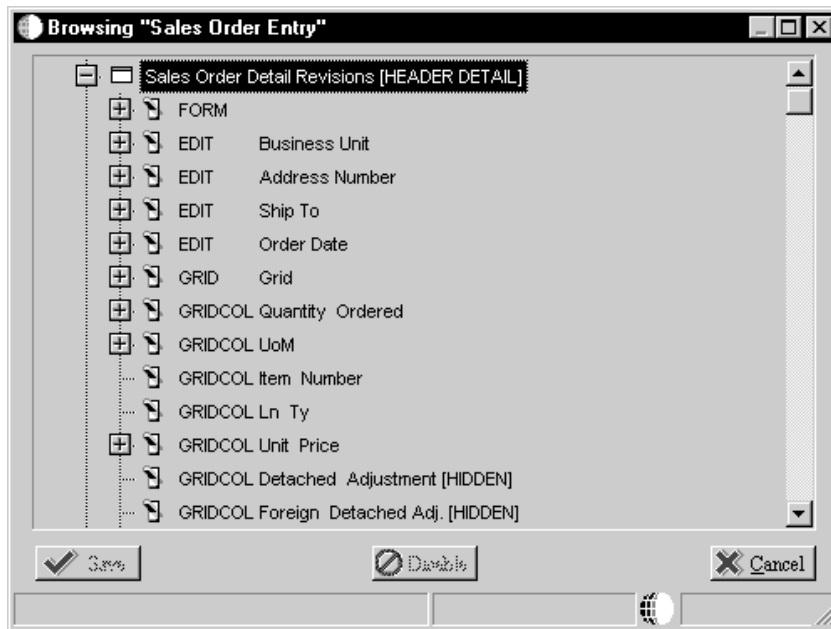
You can use BrowsER to disable or enable event rules, then observe the impact on your application.

To work with BrowsER

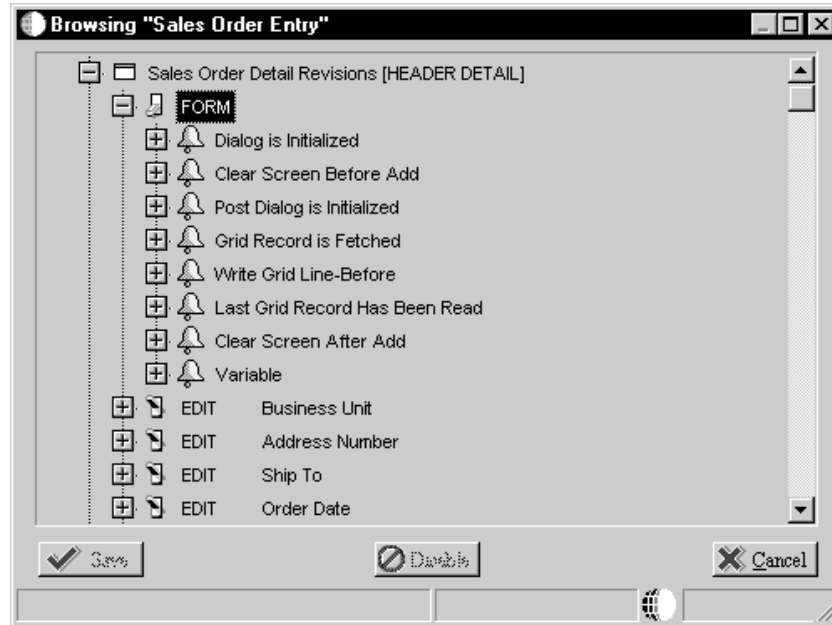
1. On Object Librarian Interactive Design, click the Design Tools tab, and then click *Browse Event Rules*.



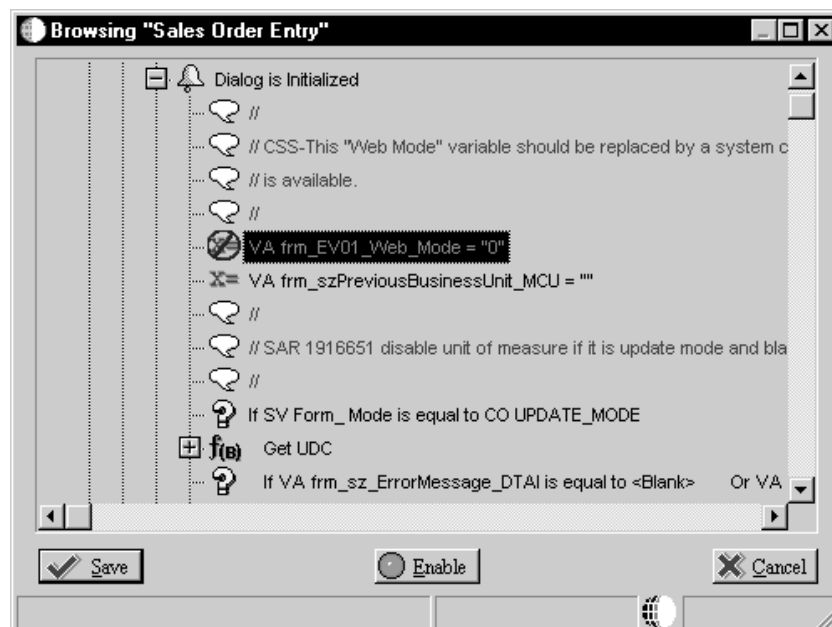
2. On BrowsER, click the plus (+) and minus (-) buttons to expand or collapse the hierarchical view of events for interactive forms or batch report sections.



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.

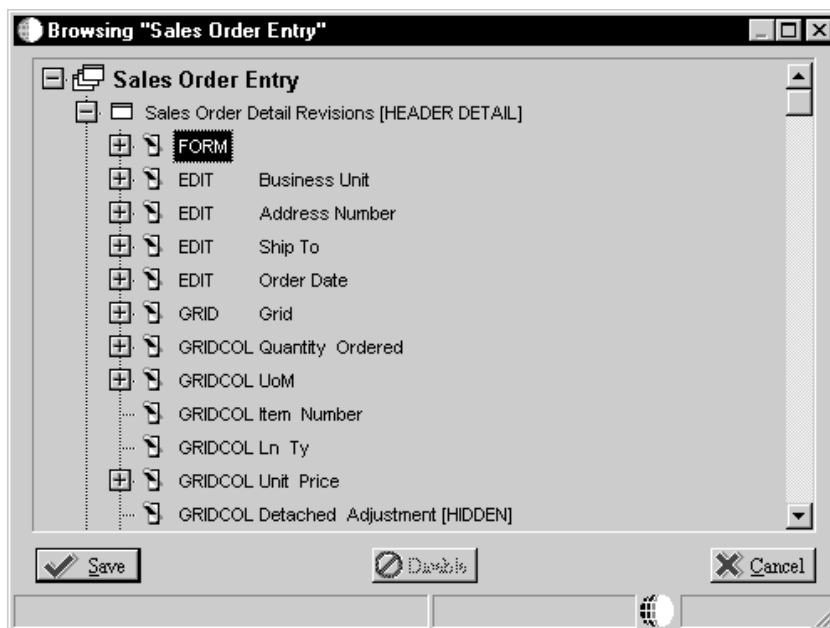
Working with BrowsER Options

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

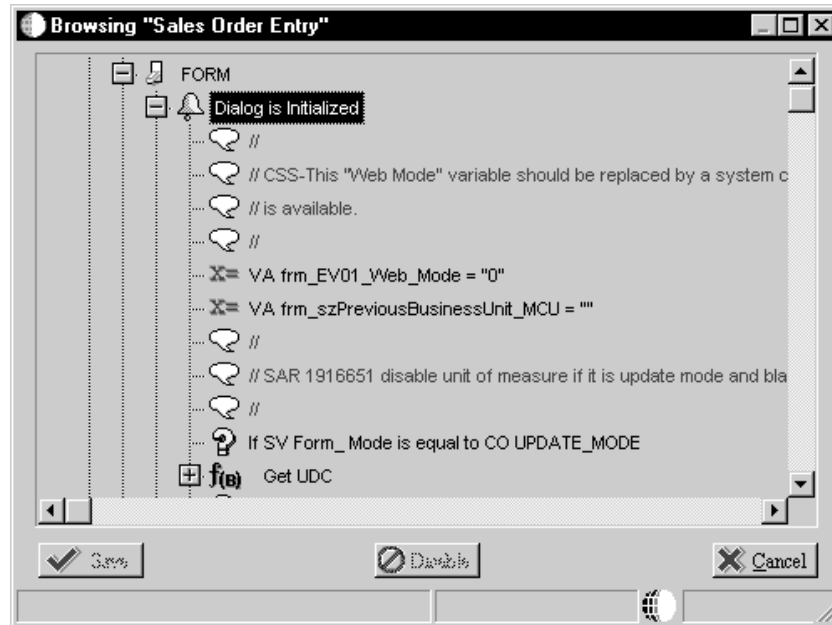
► **To work with BrowsER options**

On the Browsing form, right-click and choose an option from the menu that appears.



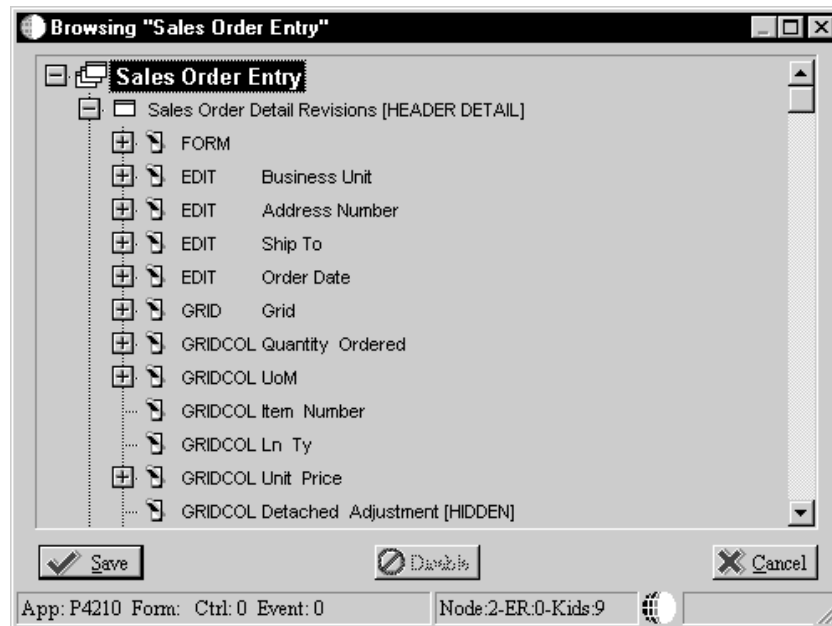
Expand Node

Expand Node allows you to expand a highlighted node to show each line of code for the node.



Show Object IDs

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.

Using Visual ER Compare

Using Visual ER Compare

Visual ER Compare is a utility that lets you compare ER on your local workstation to ER in the central objects data source of any defined pathcode, TAM specifications, or ESU backup. For example, if you make changes to the ER for an application and then want to compare your changes to the ER in the server application, you would use Visual ER Compare.

Visual ER Compare provides a line-by-line, on-screen comparison. You can change the target ER (your local version) within the utility by moving lines directly from the source ER. You can also remove or disable lines. In addition to providing an on-screen comparison, you can choose to print a report detailing the changes as well.

Using Visual ER Compare is composed of the following topics:

- Launching Visual ER Compare
- Understanding the Visual ER Compare interface
- Working with Visual ER Compare

Launching Visual ER Compare

Apply the following task only to objects with attachable ER (applications, UBEs, tables, and business functions).

To launch Visual ER Compare

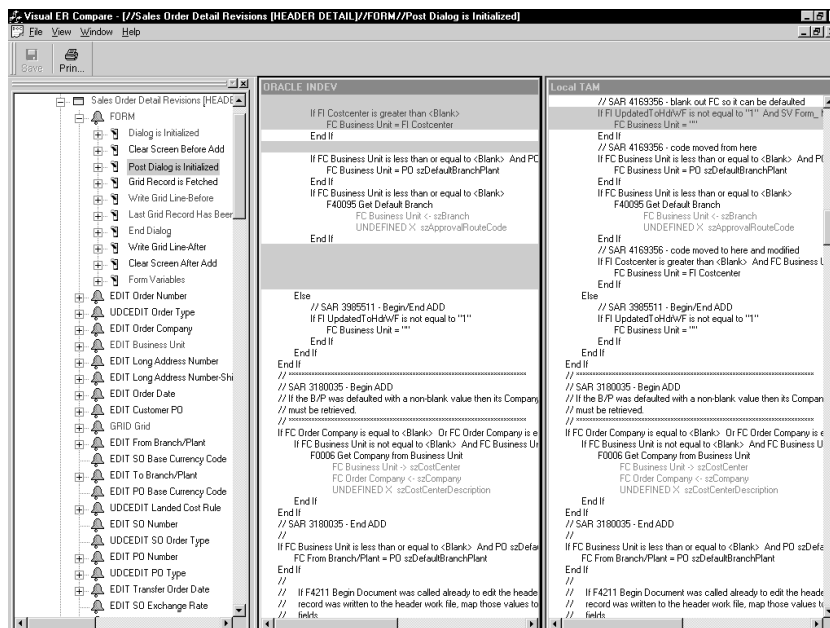
From Cross Application Development Tools (GH902), choose Object Management Workbench (P98220).

1. On Object Management Workbench, check out an object.
2. Select the object you checked out, and then click the Design button in the center column.
3. On the Design form, click the Design Tools tab.
4. Click Visual ER Merge.

5. On Select the Location of Source Specifications, perform one of the following actions:
 - Click Select PathCode, and then enter the server location of the source object (the object to which you want to compare your local ER).
 - Click Advanced TAM, and then enter the TAM location of the source object (the object to which you want to compare your local ER). ESUs are delivered in a TAM package, so use this method to compare your local ER to an object packaged in an ESU.

Understanding the Visual ER Compare Interface

When you launch Visual ER Compare, the Visual ER Compare form appears. A tree-structured menu of the ER appears on the left. The rest of the form displays the source ER (in the middle panel) and the target ER (in the right panel). The target ER is your local ER.



In the ER menu, the system uses fonts of different colors to show the rules that exist in both source and target but are different and the rules that exist in one location but not the other. The system indicates a change in the parent node if one or more of its children have been changed. In the ER panes, the system highlights the lines that differ. If lines have been added to or deleted from one side, blank lines appear on the other. Disabled lines are indicated with an exclamation point. The system uses fonts of different colors to show the rules that have changed in content or that have been added or deleted. You can change the display colors by selecting from User Options from the View menu, and then selecting Set Colors.

Visual ER Compare uses an algorithm to compare lines to determine if a change has occurred. If a certain percentage of the target line is different from the source line, then the system marks the line as being different. You can change the sensitivity of the comparison by choosing User Options from the View menu and then choosing Comparison Factors. To include disabled ER lines in the comparison, click Disable Partial Matching for Disabled ER. To change the percentage of difference required to highlight a line as being changed, enter a number in the Partial Match Ratio field. The default value of .50 means that a minimum of 50% of the target line must vary from the source line to trigger the system to mark it as being changed.

You can choose a different source by choosing Open Source from the File menu.

Working with Visual ER Compare

Use the ER menu to identify and display specific ER components that have changed. If a parent node is identified as being changed, expand it to see which of its children are different. Double-click an event in the ER menu to display its associated code. You can display more than one event at a time. Use the Tile option in the Window menu to view different ER events simultaneously.

You can also move from change to change by right-clicking in either the source or target pane and choosing Next ER Difference to move forward or Previous ER Difference to move backwards.

You can change your target ER with Visual ER Compare. You can also print the changes. If you want to copy all of the changes from the source to the target, you can use the AutoMerge feature.

Working with Visual ER Compare is composed of the following topics:

- Changing your target ER
- Printing a Visual ER Compare report
- Using AutoMerge

Changing Your Target ER

Perform any of the following actions to change your target ER:

To copy selected lines from source to target	Select the lines to copy, right-click in the source pane, and then choose Copy Right.
To delete selected lines from the target	Select the lines to delete, right-click in the target pane, and then choose Delete.

To enable or disable selected lines in the target

Select the lines to enable or disable, right-click in the target pane, and then choose Enable/Disable ER.

Note: Use the shift key to select multiple, contiguous lines and the control key to select multiple, noncontiguous lines.

After making your changes, right-click and choose Save ER. This action saves your changes to a buffer. When you choose to open a new source or to exit Visual ER Compare, the system prompts you to save your changes again. If you elect to save your changes at this time, then the system updates the object on your workstation; otherwise, your changes will be lost.

Printing a Visual ER Compare Report

You can print a report comparing the source and target ER. You can show the comparison for a particular event or for all of the ER in the object.

To print a report for an event, double-click the event in the ER menu, right-click in either pane, and then choose Print ER.

To print a report for the entire object, choose Print ER from the File menu.

Using AutoMerge

Use AutoMerge when you want the system to change the target ER to match the source ER. You can use AutoMerge to change a particular event or to update all of the ER in the object.

Caution: Before performing an AutoMerge for an entire object, do a comparison to be certain that you really want all of the changes that the system detects.

To use AutoMerge on an event, double-click the event in the ER menu, right-click in either pane, and then choose AutoMerge.

To use AutoMerge on an entire object, choose Advanced Operations from the View menu, and then choose Auto Merge.

Business Functions



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
- Understanding Business Function Processing Failover

Business Function Features

Feature	Explanation
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 connection 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 Management Workbench Business Function Source Librarian and Business Function Design
Data Structure Specifications	Object Management Workbench Business Function Parameter Design
.C file	Generated in Business Function Design Modified with the IDE
.H file	Generated in Business Function Design Modified with the IDE

The DLLs 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 where 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
CBUSPART	Business Partner
CCONVERT	Conversion Business Functions
CCORE	Core Business Functions
CCRIN	Cross Industry Application
CDBASE	Tools – Database
CDDICT	Tools – Data Dictionary
CDESIGN	Design Business Functions
CDIST	Distribution
CFIN	Financials
CHRM	Human Resources
CINSTALL	Tools Install
CINV	Inventory
CLOC	Localization
CLOG	Logistics Functions
CMFG	Manufacturing
CMFG1	Manufacturing – Modified BFs
CMFGBASE	Manufacturing Base Functions

DLL Name	Functional Group
COBJLIB	Tools – Object Librarian
COBLIB	Busbuild Functions
COPBASE	Distribution/Logistic Base Functions
CRES	Resource Scheduling
CRUNTIME	Tools – Run Time
CSALES	Sales Order
CTOOL	Tools – Design Tools
CTRAN	Transportation
CTRANS	Tools – Translations
CWARE	Warehouse
CWRKFLOW	Tools – Workflow
JDBTRG1	Table Trigger Library 1
JDBTRG2	Table Trigger Library 2
JDBTRG3	Table Trigger Library 3
JDBTRG4	Table Trigger Library 4
JDBTRIG	Parent DLL for Database Triggers

Do not use the table triggers for regular business functions.

How Distributed Business Functions Work

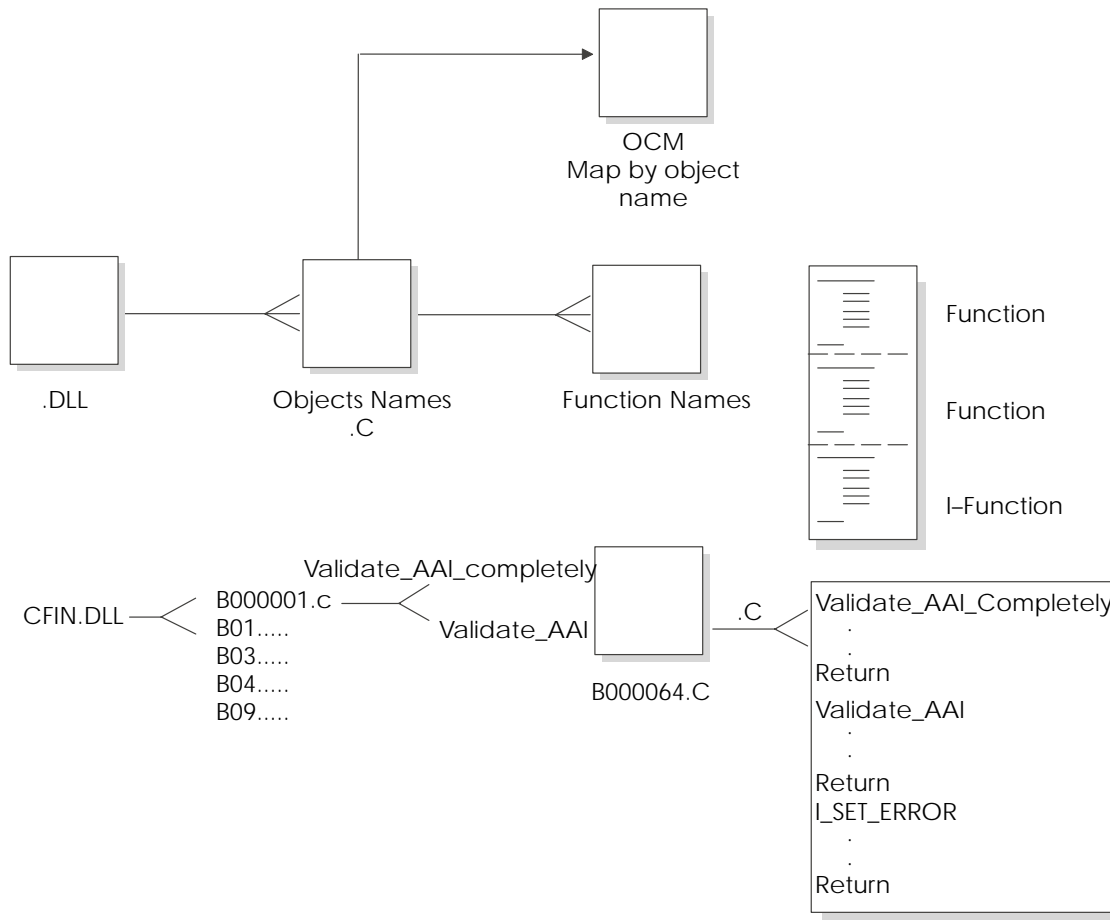
There are three main pieces that make up named event rules or business functions.

- 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**

1. On Object Management Workbench, create a Business Function.

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.

2. Choose the Parent DLL.
3. Choose the parent library in which you want this business function to reside.
4. Choose Parameters from the Row menu.
5. On Business Function Parameters, choose Select from the Form menu.
6. 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, the system identifies the business structure. You can choose another data structure or edit the one that you want to use.

► **To create or edit event rules for a business function event rule**

1. On Business Function Design, from the Form menu, choose Edit to create or edit the event rules that 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

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:
// SO 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 CO SUCCESS

VA evt_szChildATSQ = substr([VA evt_szTempATSQ],0,[VA
evt_mnSequencePosition])
If VA evt_szChildATSQ is equal to VA evt_szParentATSQ
F32943.FetchSingle
If SV File_IO_Status                is equal to CO SUCCESS

VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Config String Format Segments Cache
End If
End If
F3296.FetchNext
End While
F32943.Close
//
// Unload segments cache into the work order generic text. B3200600 Mode 6
Config String Format Segments Cache
//
End If
End If
F3296.Close
//
End If
Else
// The related sales order number is invalid. Return an error.
If BF cSuppressErrorMessages is not equal to "1"
Set NER Error("0002", BF szRelatedSalesOrderNumber)

```

```
End If  
End If
```


Understanding C Business Functions

This section includes information about:

- Understanding header file sections
- Understanding the structure of a business function source file

Understanding Header File Sections

The following are the major sections of a business function header file:

Section	What it Includes	Description
Header File Comment	<ul style="list-style-type: none"> • Header file name • Description • History • Programmer • Software Action Request (SAR) Number • Copyright information 	<p>Built by the input process of the Business Function Source Librarian.</p> <p>The programmer name and SAR number are manually updated by the programmer.</p>
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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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

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

#if defined (WIN32)
    #if defined (b98sa001_c)
        #define JDEBFRTN(r) __declspec(dllexport) r
    #else
        #define JDEBFRTN(r) __declspec(dllimport) r
    #endif
#else
    #define JDEBFRTN(r) r
#endif

/*****
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	<p>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:</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding-left: 20px;">Comment Block</td> <td>Describes the data structure</td> </tr> <tr> <td style="padding-left: 20px;">Preprocessor Directives</td> <td>Ensures the data type is not defined more than once</td> </tr> <tr> <td style="padding-left: 20px;">Typedef</td> <td>Defines the new data type</td> </tr> <tr> <td style="padding-left: 20px;">#define</td> <td>Contains the ID to be used in processing if the related data structure element is in error</td> </tr> <tr> <td style="padding-left: 20px;">#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	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	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	<p>Business Function Standard</p> <p>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.</p> <p>Parameters:</p> <p>LPBHVRCOM: Pointer to a data structure used for communication with business functions. Values include an environment handle.</p> <p>LPVOID: Pointer to a void data structure. Currently used for error processing and will be used for security in the future.</p> <p>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.</p> <p>Parameter names (lpBhvrCom, lpVoid, and lpDS) will be the same for all business functions.</p> <p>JDEBFRTN(ID)JDEBFWINAPI: All business functions will be declared with this return type. It ensures that they are exported and imported properly.</p>										
13.	Business Function Design	Internal function prototypes required to support the business functions in this source file										

Understanding the Structure of a Business Function Source File

The Object Management Workbench builds a template for the business function source file. The business function source file consists of several major sections.

Section	What it Includes	Description
Source File Comment Block	<ul style="list-style-type: none"> Source file name Description History Programmer Date SAR Number Description Copyright information 	<p>Built from the information given in the Business Function Source Librarian.</p> <p>The programmer name and SAR number are manually updated by the programmer.</p>
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	<ul style="list-style-type: none"> Business function name Description Description list of the parameters 	
Business Function Source Code	Source code for the business function	
Internal Function Comment Block	<ul style="list-style-type: none"> Function name Notes Returns Parameters 	<p>Should be copied and values placed in the specified sections to describe the internal function. Internal function source code must follow the comment block.</p>
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.
*****/
/*****
*   Notes:
*****/
4. #include <B98SA001.h>
/*****
*   Business Function:  CheckForInAddMode
*
5. *   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
*****/
6. JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid,
LPDS104438 lpDS)
{
7. /*****
*   Variable declarations
*****/
8. /*****
*   Declare structures
*****/
9. /*****
*   Declare pointers
*****/
10. /*****
*   Check for NULL pointers
*****/
if ((lpBhvrCom == NULL) ||
    (lpVoid == NULL) ||
    (lpDS == NULL))
11. {
    jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", 0);
    return ER_ERROR;
}
12. /*****
*   Set pointers
*****/
13. /*****
*   Main Processing
*****/

```

```

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. Several categories of APIs exist, 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 application programming interfaces contains the following topics:

- Using common library APIs
- Using database APIs
- Calling an API from an external business function
- Calling a Visual Basic program from OneWorld

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 by 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 provides two main data types that you should be concerned with when you create business functions. They are:

- MATH_NUMERIC
- 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

Using database APIs contains the following topics:

- Understanding standards and portability
- J.D. Edwards open database connectivity (ODBC)
- Standard JDEBASE API categories
- Connecting to a database
- Understanding database communication steps

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).

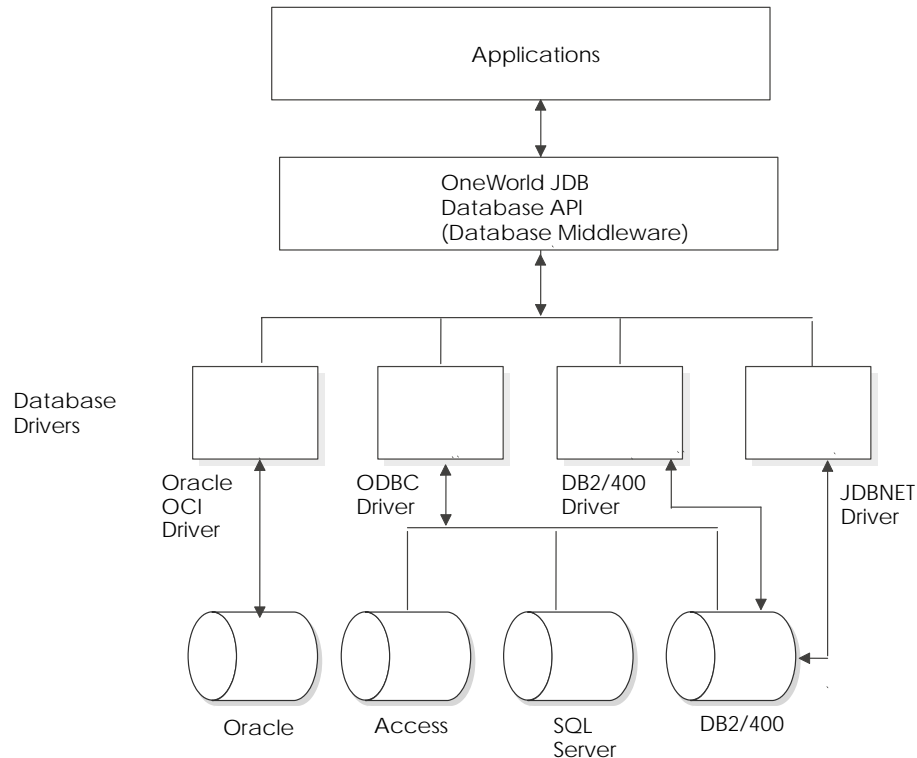
J.D. Edwards Open Database Connectivity (ODBC)

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. The database driver is responsible for processing the API request and communicating 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: <ul style="list-style-type: none"> • 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

Categories	What it Does
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

API Level	Communication Handles	API Name
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, 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.

Examine the following calling conventions:

- Stdcall
- Cdecl

Stdcall Calling Convention

The following code example is standard code for windows programs. It is not OneWorld specific.

```

# 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 line
        // 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

```

NOTE: These calls will only work on a windows client machine. LoadLibrary & GetProcAddress are Windows APIs. If the business function is compiled on a server, the compile will fail.

Calling a Visual Basic Program from 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 using the following process:

- You must write the Visual Basic program into a Visual Basic DLL that exports the function name of the program and returns a parameter 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 that you create, call the win32 function GetProcAddress to get the Visual Basic function and call it.

Working with Business Functions

Every business function must follow a defined structure and form. Every line of code must conform to the 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 Management Workbench to build business function data structures.
- Use Object Management Workbench to create business function source and header files.
- Create and add data structure type definitions to the header file.

This chapter describes:

- Launching Business Function Design
- Viewing Object Codes
- Adding Related Tables and Functions
- Debugging Business Functions
- Changing a Business Function Parent DLL

See Also

- *Creating a Business Function Data Structure*

Launching Business Function Design

If you have just created a new business function, skip to step 3.



To launch Business Function Design

1. On Object Management Workbench, check out the business function with which you want to work.
2. Ensure the business function is highlighted, and then click the Design button in the center column.

The Business Function Design form appears.

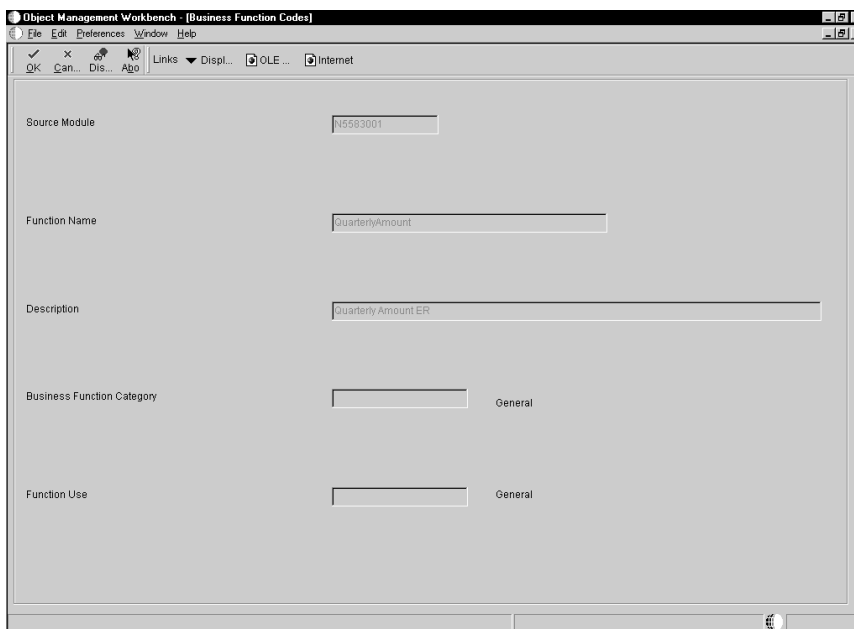
3. Click the Design Tools tab, and then click *Start Business Function Design Aid*.

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**

1. From Business Function Design, 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**

1. On Business Function Design, 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**

1. From Business Function Design, 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**

1. On Business Function Design, type the new DLL name in Parent DLL, or change the business function location and click OK.
2. On Object Librarian Business Function Design, click OK.
3. On Object Management Workbench, 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 process 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

1. On Object Management Workbench, click Add.
2. On Add OneWorld Object to the Project, choose the Business Function Library option, and then click OK.
3. On Add Object, complete the following fields and click OK.
 - Object Name
 - Description
 - Product Code
 - Product System Code
 - Object Use

Specifying a Custom DLL for a Custom Business Function

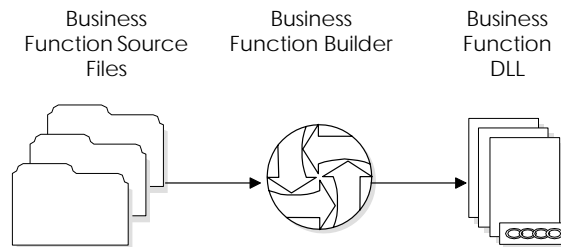
When you create a custom business function, you need to specify one of your custom DLLs; otherwise, the build process builds the custom business function into the J.D. Edwards CCUSTOM.DLL, which is the default. Use this task to specify a custom DLL whenever you create a custom business function.

► **To specify a custom DLL for a custom business function**

1. 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.)
2. Run the build for the business function.

Working with Business Function Builder

You use Business Function Builder to build business function code into a dynamic link library (DLL). You can build C business functions, named event rules, and table event rules.



The process that occurs when you 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
- Building all business functions
- Reviewing error output
- Resolving errors

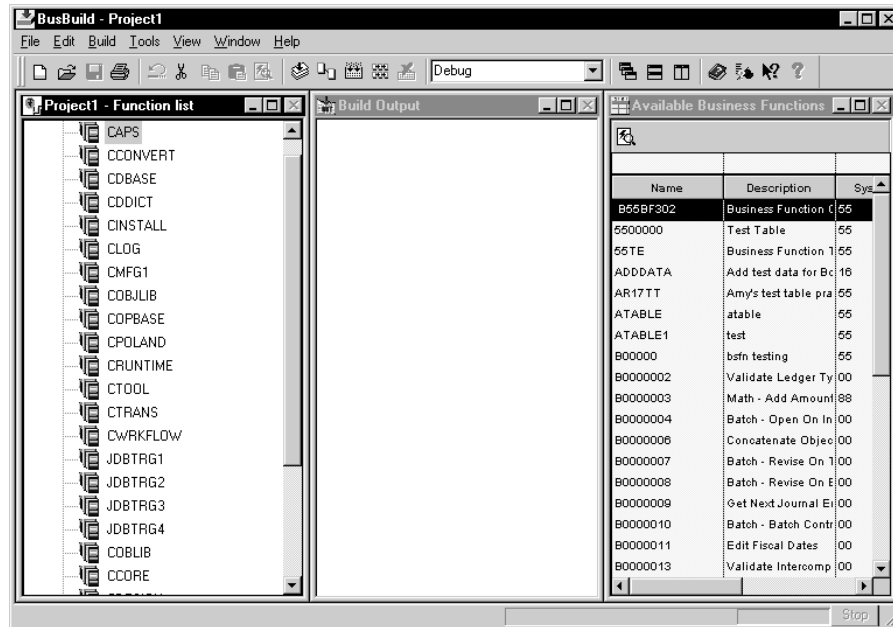
Understanding the Business Function Builder Form

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 that you select to link or compile during a build. Any functions that 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 that you can see the build in progress. This display is useful for diagnosing any problems that might 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.

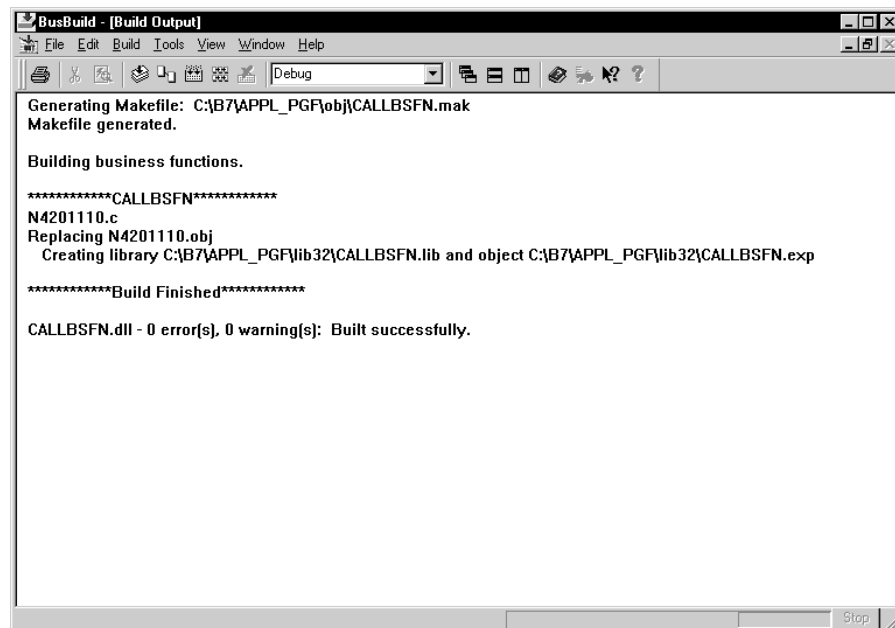
Building a Single Business Function

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

1. On Object Management Workbench, choose the business function that you wish to build, and then click the Design button in the center column.
2. On Business Function Design, click the Design Tools tab, and then click *Busbuild Standalone*.

The BusBuild form appears and Business Function Builder begins building your business function.



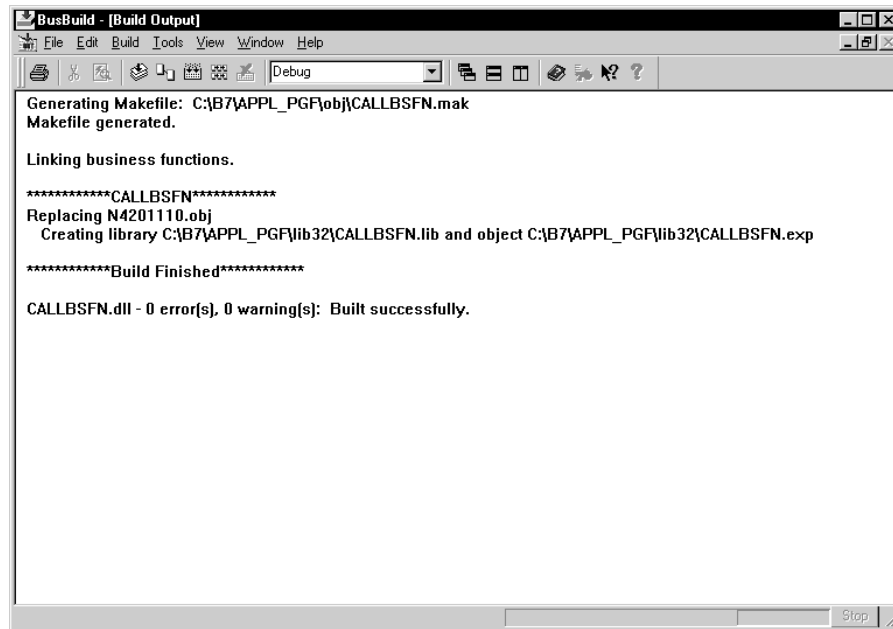
Build Output displays the results of the build. When the build is finished, the message “*****Build Finished*****” appears at the bottom of Build Output. The text following this line indicates whether the build was successful. If the build was successful, you can test your business function. Otherwise, you must correct any problems and 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 its original form.

Linking Business Function Objects

When you install one or more OneWorld applications from Object Librarian onto 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

1. On BusBuild, 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

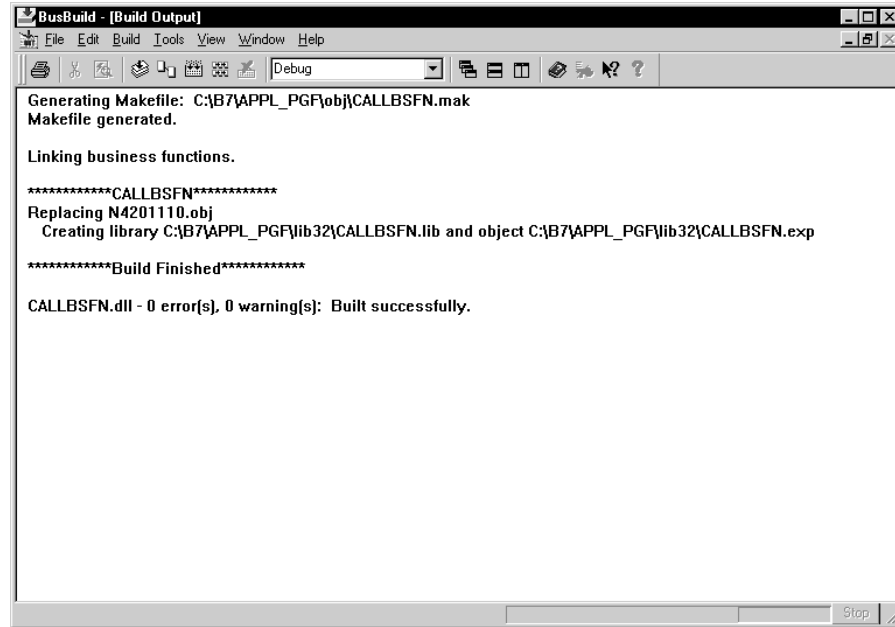
1. On BusBuild, choose one of the following options from the Tool menu:

Synchronize JDEBLC	You can run this utility to reorganize OneWorld business functions into new DLL groupings. This utility synchronizes the local JDEBLC parent specification table's 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 that you are trying to link is incorrectly compiled with PDB information. To resolve this problem, you can use the PDB scan to identify any object fields that were built with PDB information. Recompile any business functions that the PDB scan reports.

Reading Build Output

Build Output consists of a series of sections that display important information about the status of a build. You can use this output to determine whether your build has completed successfully and to troubleshoot problems if errors occur during your build.

The following figure is an example of the messages the system generates during a 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 that 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 that 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 that 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 that you attempted to build. This report includes:

- The number of warnings
- The number of errors
- Whether or not the DLL build is successful

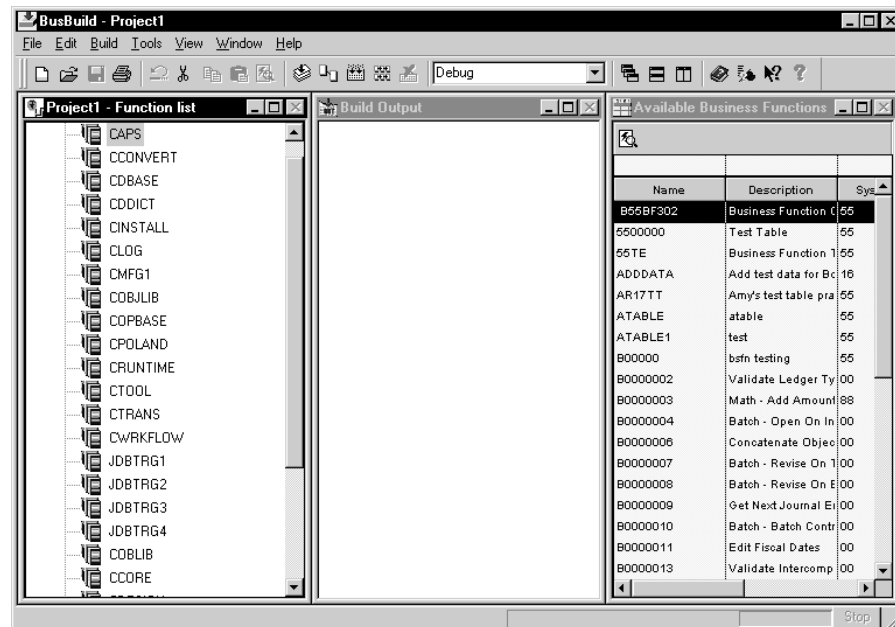
Building All Business Functions

Build All is usually done by a system administrator. Build All processes 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.

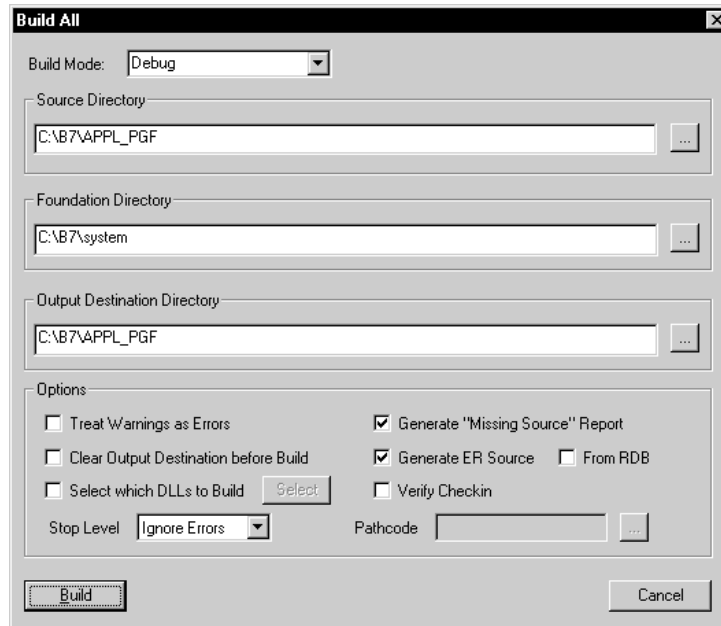
► To build all business functions

1. On Object Management Workbench, choose the business function that you wish to build, and then click the Design button in the center column.
2. On Business Function Design, click the Design Tools tab, and then click *Busbuild Standalone*.

The BusBuild form appears.



- 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.

- 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 that 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, 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.

6. Complete the Foundation Directory field.

Use this field to specify the foundation to use for this build. The foundation that 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

7. 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, 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.

8. 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 at which the build will stop. 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.

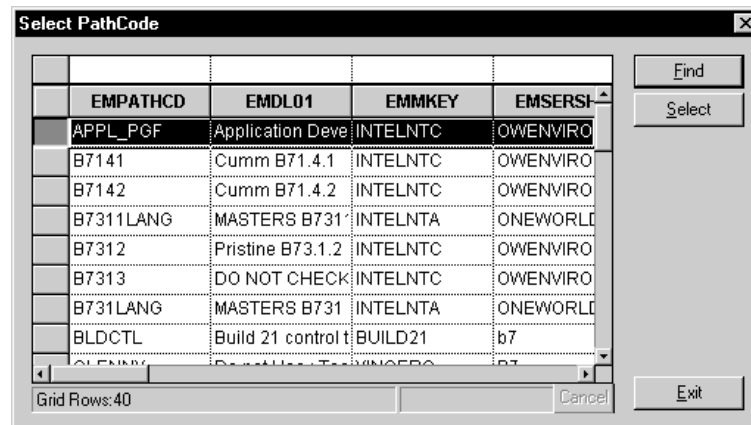
Generate ER Source

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 Check-in

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.



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.

Field	Explanation
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.
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.

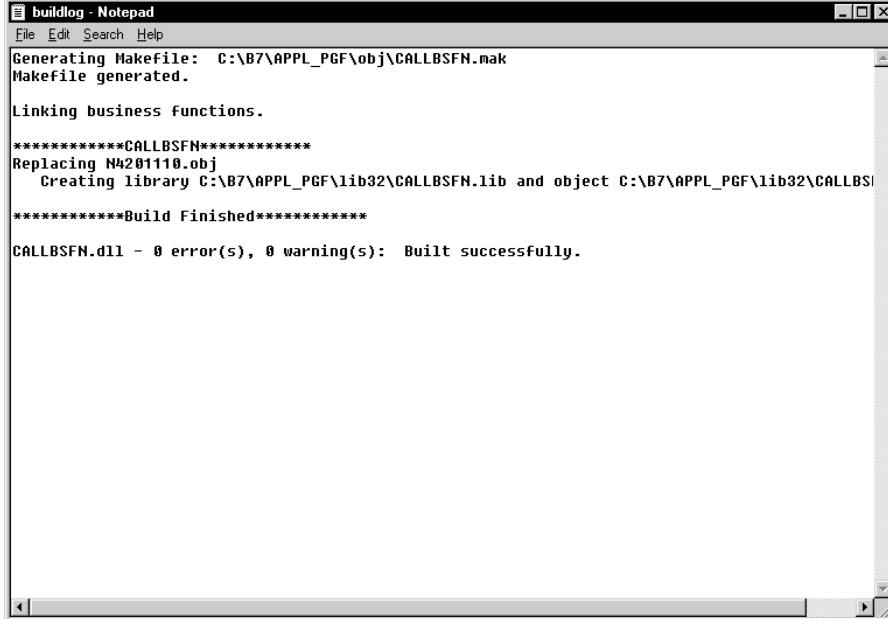
Reviewing Error Output

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.nak
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 that 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 cxxxx". 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:

- Master business function naming convention
- Creating processing options for a master business function
- Naming transaction master business function modules
- Components of a transaction master business function
- Cache structure
- Building transaction master business functions
- 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 DxxxxxxxxA,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 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 Txxxxxxxx (replace the B or XT in the MBF name with a T).

See *Creating Processing Options Data Structures* for detailed instructions on creating a processing option template.

2. Create a dummy application for attaching the MBF processing options. Different versions of the MBF processing options can then be set up through versions list. The naming convention for this dummy application is Pxxxxxxxx (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 Document	Called when all header information has been entered <ul style="list-style-type: none"> • Creates initial header if not already created. Can also include defaulting, editing, and processing options.
Edit Line	Called when all line information has been entered <ul style="list-style-type: none"> • Creates cache for detail information if not already created.
Edit Document	Called when ready to commit the transaction <ul style="list-style-type: none"> • Processes any last document edits and verifies that all records are valid to commit.
End Document	Called when you need to commit the transaction <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • Deletes header and detail cache records.

Begin Document

Function Name

xxxxxxBeginDocument

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

- You must 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 or deleting detail records, 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; otherwise, pass zeros and assign a job number.
Document Action	ACTN	I	A or 1 = Add C or 2 = Change D = Delete This is the action of the entire Document, not the individual detail lines. For example, you might modify a few detail lines in Edit Line, add a few detail lines in Edit Line and delete a few detail lines in Edit Line, but the <i>Document Action</i> in Begin Document would be Change.
Process Edits	EV01	I	<i>Optional</i> 0 = No Edits Any Other = Full Edits Note: 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, 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.
Name	Alias	I/O	Description
Header Field Two	****	I/O	
.	****	I/O	
.			
.			
Header Field XX	****	I/O	

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, performs calculations, and retrieves 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 Fetched* 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 a 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 gird record fields. Edit Line will provide validation, Data Dictionary edits, UDC editing, defaults, and so on.
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. These could be flags for processing, dates to validate, and so on.
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
- Example: processing all records to ensure that percentages total 100%
- Example: validating that the last record can't contain certain information

Special Logic/Processing Required

- Depending on the type of document being processed, different logic is executed. An example is 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 are 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

xxxxxxEndDocument

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 passes all edits before running this event.
- Reads cache records.
- On an ADD, writes new rows to the table.
- On a CHG, retrieves and updates existing rows.
- On a DEL, deletes rows from the table.
- Adds 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 are 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. Reads 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 <code>BeginDoc</code> .
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 line is blank, the system begins 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 line is blank, the system deletes to the end of the cache.

Cancel Document (optional)**Function Name**

xxxxxxCancelDoc

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 <code>BeginDoc</code> .

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:

Header

Field Description	Field	Key	Type	Size
Job Number	JOBS	X	Num	
Document Action	ACTN		Char	1
Processing Options				
Currency Flag	CYCR		Char	1
Business View Fields				
Work Fields				

Job Number

A unique 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, and 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

The fields required for processing the transaction, and 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

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	JOBS	X	Char	8
Line Number	(Application Specific)	X	Num	
Line Action	ACTN		Char	1
Business View Fields				
Work Fields				

Job Number

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.

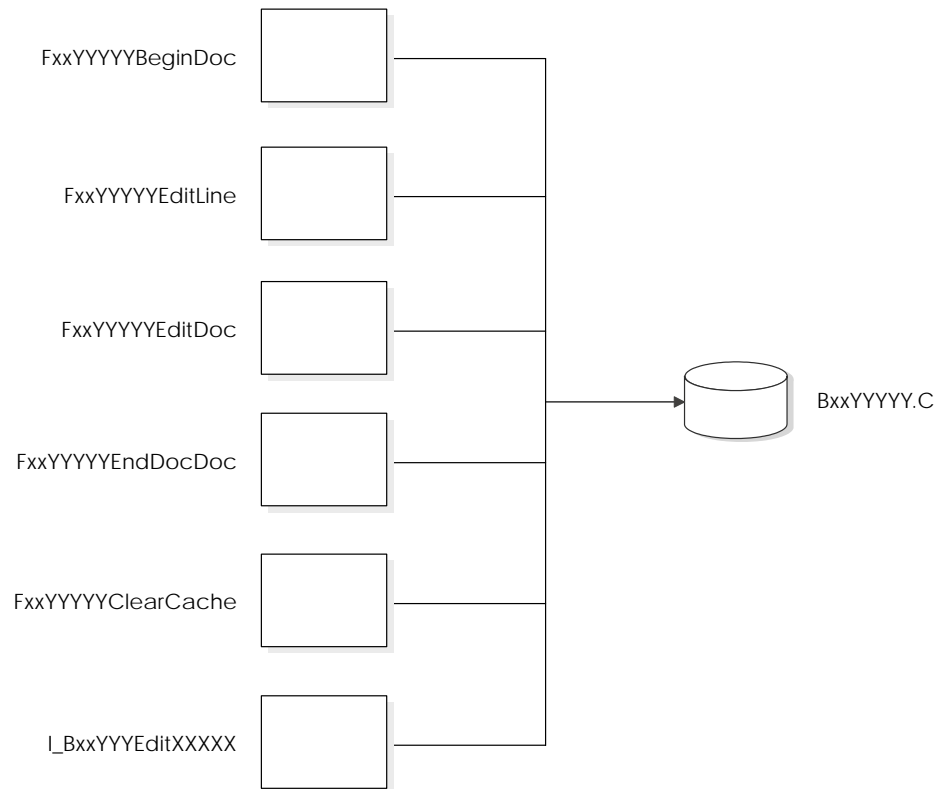
Line Number

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.

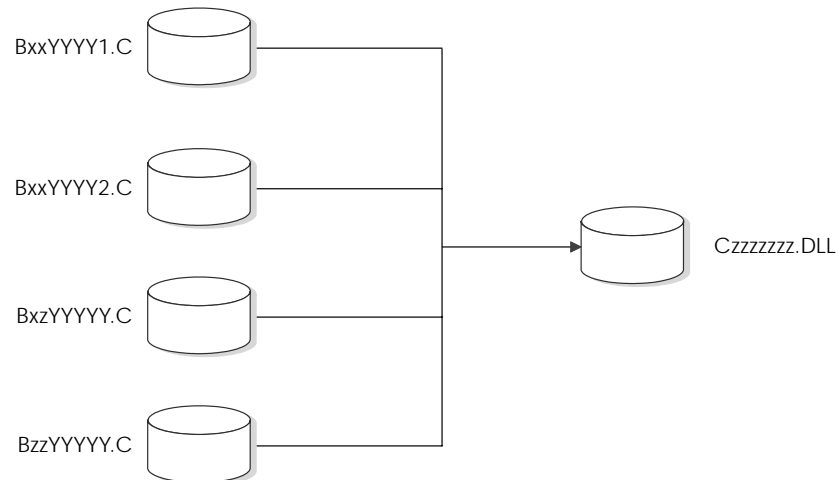
Line Action	The action for the transaction line A or 1 = Add C or 2 = Change D = Delete
Business View Fields	Fields required for processing the transaction that will be written to the database. All fields in the record format that are not saved in the detail cache will be initialized when the record is added to database using the API's.
Work Fields	Fields that are not part of the business view, but are needed for editing and updating the transaction line.

Building Transaction Master Business Functions

The following 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 reusable.
- 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 that 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. Several standards 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.

Certain circumstances might 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 can 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 processes the base data, and the F0301ProcessBillingData updates additional data.

- Subordinate detail files

Information can exist in addition to the primary master file that has been normalized out to allow for a one-to-many relationship. Designing the Master File MBF strictly on the basis of how the database is designed translates into three calls. Including at least one occurrence of a detail relationship in the data structure of a Master File MBF is valid. This inclusion allows users to establish reasonably complete master file information using a simple interface as long as they have simple needs.

Street addresses and phone numbers within Address Book are a good example. It is reasonable for 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.

Master File Master Business Functions contains the following topics:

- Information structure
- Performance impact

Information Structure

Standard Parameters for Single Record Master Business Functions

Name	Alias	I/O	Required/Optional	Description
Action Code	ACTN	I	Required	A - Add, I - Inquiry, C - Change, D - Delete, S - Same as except (The record is the same except for what the user change.)
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. For all other Action codes, this parameter is required.
Same as except	AN8	I	Optional	Required for S - Action Code The record is the same except for what the user changes.

Application Parameters (Example: Address Book)

Name	Alias	I/O	Required/Optional	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*Processxxxxxxxxxx, 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 configuration 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 can choose not to apply all of the fields.

Through different interfaces, the user can 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:

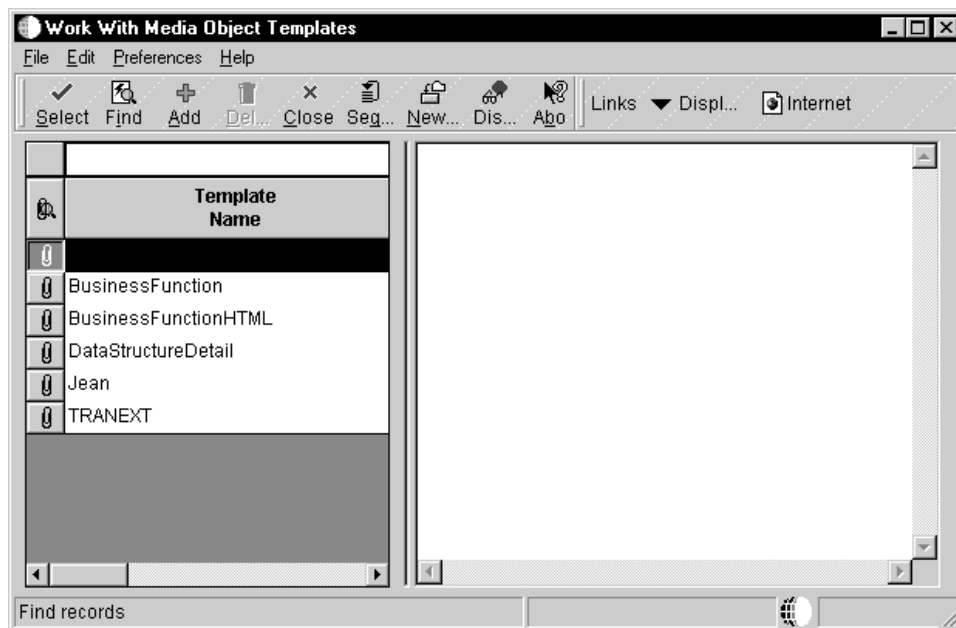
- | | |
|--------------------------------|--|
| Business Function Notes | Shows you the documentation for the specific business function you are using |
| Data Structure Notes | Displays notes on the data structure for the business function |
| Parameter Notes | Displays notes on the actual parameters in the data structure |

Creating Business Function Documentation

Use the following process to create business function documentation.

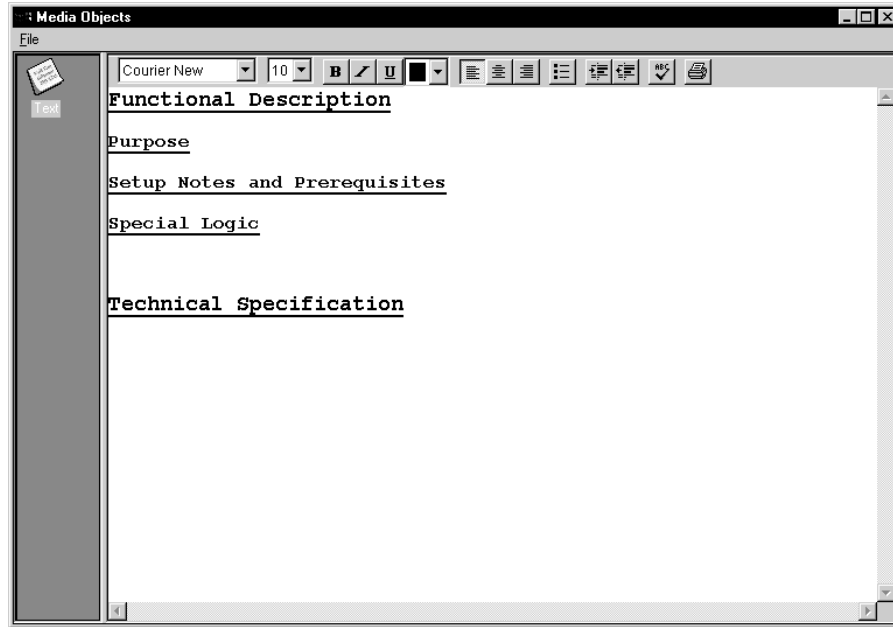
► To create business function documentation

1. On Object Management Workbench, choose the business function that you wish to document, and then click the Design button in the center column.
2. On Business Functions Design, click the Attachments tab.
3. Right-click in the icon bar on the left side of the form, and then select Templates.
4. On Work With Media Object Templates, click Find.



5. Select the template that you want to use.

J.D. Edwards uses the Business Function template as a standard.



6. Type in the appropriate information under each template heading.

Business Function Documentation Template

The Business Function documentation template contains the following sections:

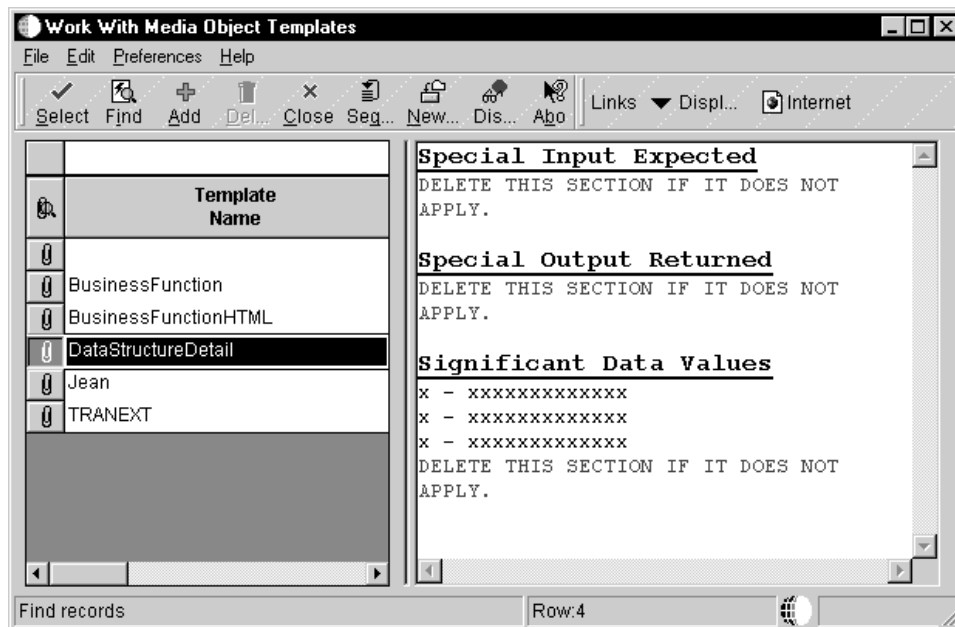
Purpose	This section contains a brief summary of what the function does.
Setup Notes and Prerequisites	This section includes any special notes to assist in using the function, including prerequisite functions, special values that need to be initialized, events recommended to run the function, or if memory 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. It can be a direct copy from an existing word processing document.

Creating Data Structure Documentation

Use the following process to create data structure documentation.

► **To create data structure documentation**

1. On Object Management Workbench, check out the data structure that you want to document.
2. Ensure the data structure is highlighted, and then click the Design button in the center column.
3. On Data Structure Design, click the Attachments tab.
4. Right-click in the icon bar on the left side of the form, and then select Templates.
5. On Work With Media Object Templates, click Find.



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, with one exception. After selecting the data item in the structure for which you want to enter notes, click the Data Structure Item Attachments binder clip. Parameter documentation has no special template.

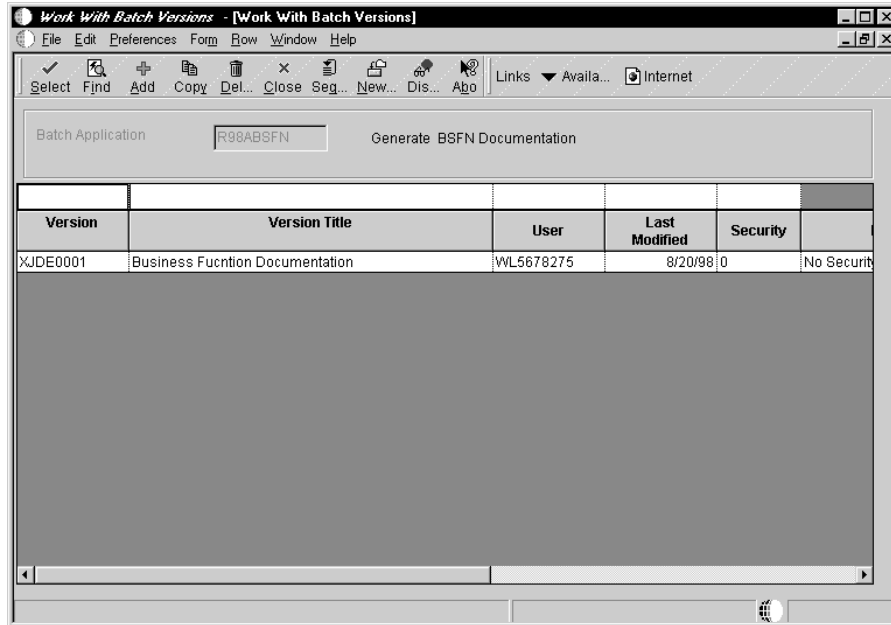
You can enter specific notes about a data structure item to further clarify the information that should be passed in or passed out of the item—for example a mode parameter. The notes should indicate the valid values that the function will accept when you hook it up and how to use them. For example, 1 = Add mode, or 2 = Delete.

Generating Business Function Documentation

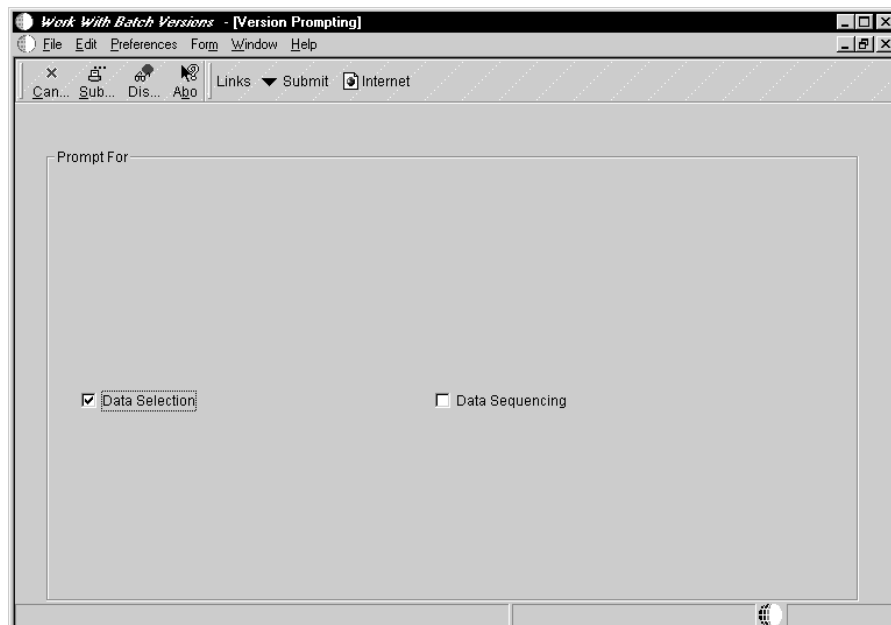
Generating business function documentation provides you with an online list of business function documentation that allows you to view documentation through the Business Function Documentation Viewer (P98ABSFN). Typically the system administrator performs this task because generating the business function documentation for all business functions takes a long time. If you create new business function documentation, you need to regenerate the business function documentation just for that business function.

To generate business function documentation

1. From the Cross Application Development Tools menu (GH902), select Generate BSFN Documentation.



2. On Work with Batch Versions, choose version XJDE0001.

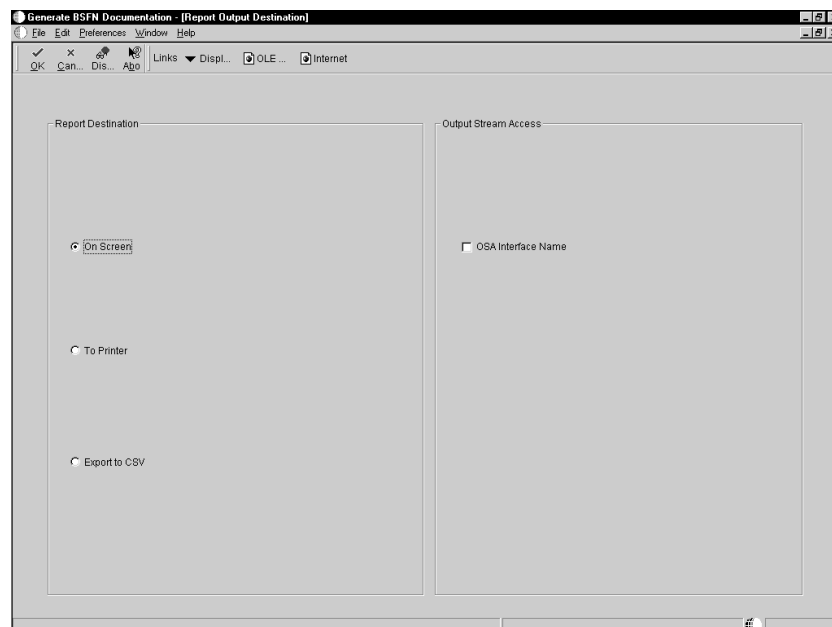


3. If you do not want to generate all business function documentation, on Version Prompting, choose the following option:
 - Data Selection
4. Click Submit.
5. On Data Selection, build your criteria for data selection, and then click OK.

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

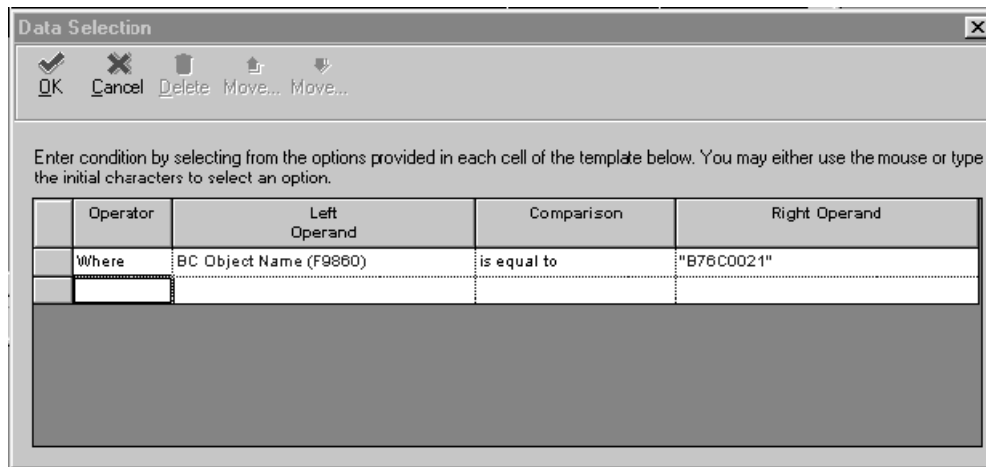
A hypertext markup language (HTML) link is created for each business function for which you generated documentation. An Index HTML file is also created. These HTML files are placed in your output queue directory. Output is in the following format:

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

Data Selection Tips

You can use data selection to choose the business functions for which you wish to generate documentation. R98ABSFN uses your data selection criteria to filter the business function documentation. It takes longer to run when you generate documentation you do not need. If you generate documentation for all business functions, the process can take quite a while. You can use data selection to generate documentation for one business function, all business functions, or any combination between.

For example, if you want to generate documentation for a single business function you can use the data item BC Object Name (F9860).



If you want to generate documentation for all of the business functions for a specific product code, such as Payroll, you use the data item BC Product Code (F9860).

	Operator	Left Operand	Comparison	Right Operand
	Where	BC Product Code (F9860)	is equal to	"07"

You can also use the right operand on the Data Selection form to choose ranges or lists of values to further refine your filter.

You can filter using any value that is associated with a business function. For example, you can use BC Date – Updated (F9860) if you have already produced the documentation for a previous release of OneWorld and you want only new or modified business function documentation after an upgrade or update of OneWorld.

You use BC Function Type (F9860) to choose Master business function documentation.

You use BC Location Business Function (F9860) to produce documentation for client run business functions.

You use BC Object Type (F9860) to generate documentation for NERs only.

You can use many other informational fields to choose the business functions for which you wish to generate documentation.

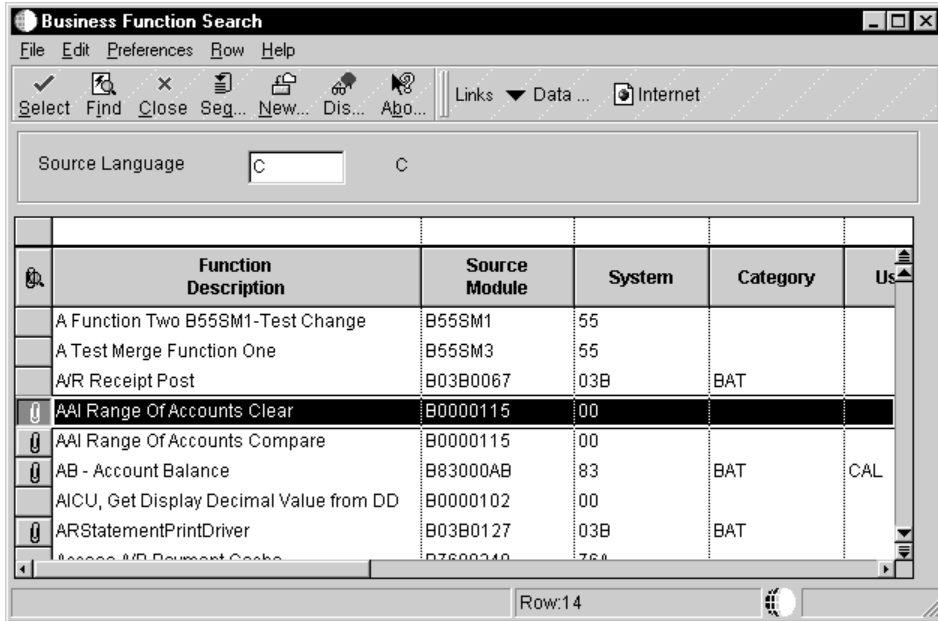
Viewing Business Function Documentation

You can view your business function notes from several different locations, including:

- Business Function Search
- Business Function - Values to Pass
- Business Function Documentation Viewer

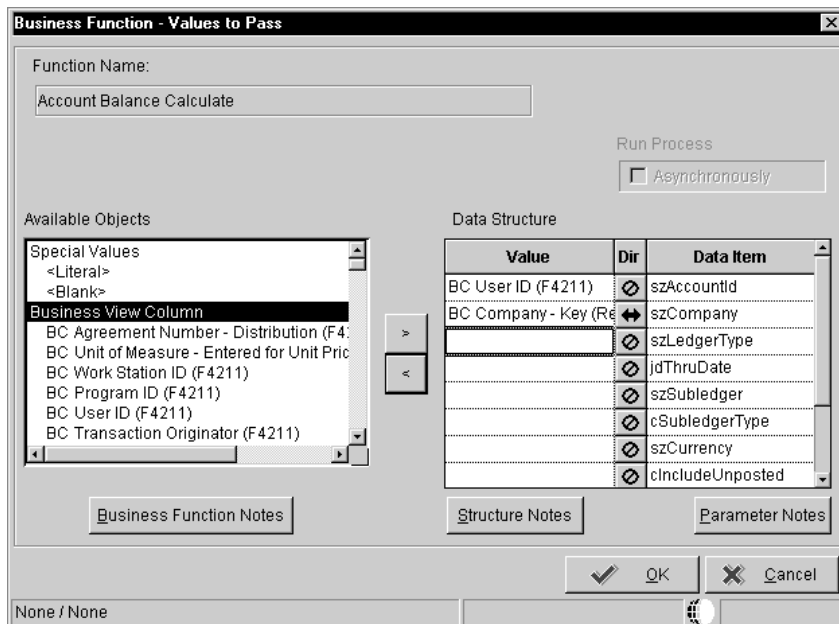
Viewing Documentation from Business Function Search

When you make a connection to a business function in event rules, the Business Function Search form appears.



You can then select the function that you want to call. From the row menu, choose Data Structure Notes or Attachments to view the documentation for the business function.

Viewing Documentation from Business Function - Values to Pass



You can click 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

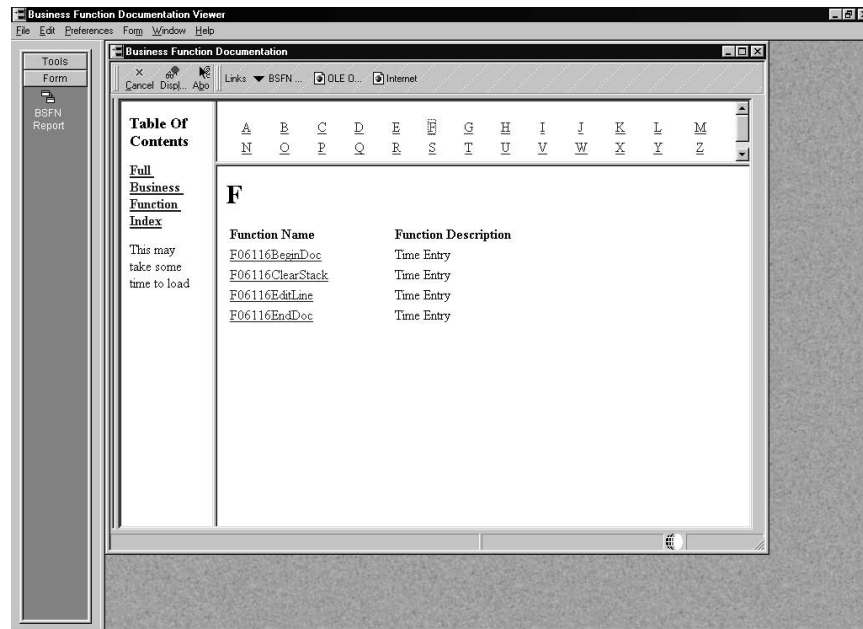
BSFN Notes Displays the notes for the business function.

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.

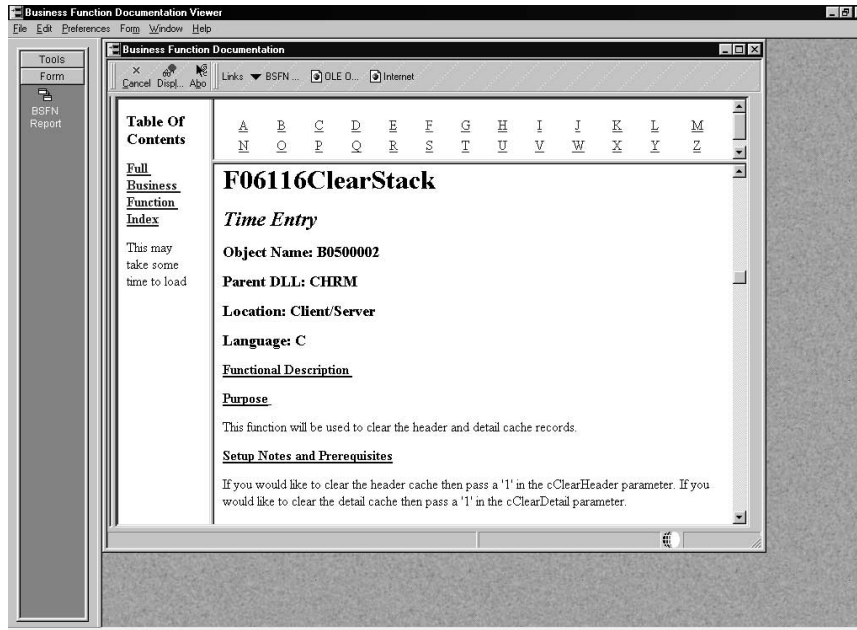


After you have generated your report, use menu GH902 to access the Business Function Documentation Viewer (P98ABSFN) to display your information. J.D. Edwards suggests that you use this method to view business function documentation.

The Business Function Documentation form contains the HTML index that 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 for all business functions. Because the generation process places the documentation files in the local directory, the administrator must then copy the files to a central directory on the deployment server. The files must be copied to the media object queue for media object business function notes. If you are 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 in the print queue directory in your OneWorld directory.

Understanding Business Function Processing Failovers

In some instances where a business function fails to process correctly, OneWorld can attempt to recover and reprocess the transaction. The system recognizes two principle failure states: process failure and system failure.

A process failure occurs when a jdenet_k process aborts abnormally. For a process failure, OneWorld server processing launches a new jdenet_k process and continues processing.

A system failure occurs when all OneWorld server processing fails, the machine itself is down, or the client cannot reach the server due to network problems. For a system failure, business function processing must be rerouted either to a secondary server or to the local client. The system uses the following process to attempt to recover from this state.

When the call to the server fails, the system attempts to reconnect to the server.

- If reconnect succeeds and no cache exists, the system reruns the business function on the server. If a cache does exist, the system forces the user out of the application.
- If reconnect fails and no cache exists, the system switches to a secondary server or to the local client. If a cache does exist, the system forces the user out of the application.

After one module has switched, all subsequent modules switch to the new location.

Caching



Caching

You can use caching to improve performance. There are two types of caching in One World.

- 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.
- 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:

- Understanding JDECACHE
- Working with JDECACHE
- JDECACHE cursors
- JDECACHE errors
- JDECACHE example program
- JDECACHE standards



Understanding 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 where 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 on which it is running. 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.

Understanding JDECACHE contains the following topics:

- When to use JDECACHE
- Performance considerations
- The JDECACHE API set

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 that 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. 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 that the user enters in one purchase order in memory. The number of records that you store depends on several things, including your cache buffer size for each record, the local memory size, and where the business function that 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.

Performance Considerations

Some guidelines for getting the best performance out of JDECACHE:

- If you use JDECACHE, cache as few records as possible.
- The fewer columns (segments) that 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.

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`
- `jdeCacheInitMultipleIndex`
- `jdeCacheInitUser`
- `jdeCacheInitMultipleIndexUser`
- `jdeCacheGetNumRecords`
- `jdeCacheGetNumCursors`
- `jdeCacheClear`
- `jdeCacheTerminate`
- `jdeCacheTerminateAll`

`jdeCacheInit/jdeCacheInitMultipleIndex` initializes the cache uniquely per user. Therefore, if a user logs in to OneWorld and then runs two sessions of the same application simultaneously, the two application sessions will share the same cache. Consequently, if the first application deletes a record from the cache, the second application will not be able to see the record. Conversely, if two users log in to OneWorld and then run the same application simultaneously, the two application sessions will have different caches. Consequently, if the first application deletes a record from its cache, the second application will still be able to see the record in its own cache.

`jdeCacheInitUser/jdeCacheInitMultipleIndexUser` initializes the cache uniquely per application. Therefore, if a user logs in to OneWorld and then runs two sessions of the same application simultaneously, the two application sessions will have different caches. Consequently, if the first application deletes a record from its cache, the second application will still be able to see the record in its own cache.

JDECACHE Manipulation APIs

You can use the JDECACHE manipulation APIs for retrieving and manipulating the data in the cache. Each API implements a cursor that acts as pointer to a record that is currently being manipulated. This cursor is essential for navigation within the cache. JDECACHE manipulation APIs should be passed handles of the following types:

HCACHE	Identifies the cache that is being worked
HJDECURSOR	Identifies the position in the cache that is being worked

The set of JDECACHE manipulation APIs consists of the following APIs:

- `jdeCacheOpenCursor`
- `jdeCacheResetCursor`
- `jdeCacheAdd`
- `jdeCacheFetch`
- `jdeCacheFetchPosition`
- `jdeCacheUpdate`
- `jdeCacheDelete`
- `jdeCacheDeleteAll`
- `jdeCacheCloseCursor`
- `jdeCacheFetchPositionByRef`
- `jdeCacheSetIndex`
- `jdeCacheGetIndex`

Working with JDECACHE

The JDB environment implicitly creates, manages, and destroys the JDECACHE environment. Each cache that is used within the JDECACHE environment is associated with a JDB user. Thus, a JDB_InitBhvr 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 6 when the actual JDECACHE APIs may be called in any order:

To call JDECACHE APIs

1. JDB_InitBhvr
2. Create index or indices
3. jdeCacheInit or jdeCacheInitMultipleIndex
4. jdeCacheAdd
5. jdeCacheOpenCursor
6. The rest of JDECACHE operations in any order:
 - jdeCacheFetch

- jdeCacheOpenCursor (the second cursor)
 - jdeCacheFetchPosition
 - jdeCacheUpdate
 - jdeCacheDelete
 - jdeCacheDeleteAll
 - jdeCacheResetCursor
 - jdeCacheCloseCursor (if the second cursor is opened)
7. jdeCacheCloseCursor
 8. jdeCacheTerminate
 9. JDB_FreeBhvr

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:

```
#define JDECM_MAX_UM_SEGMENTS 25
struct _JDECMKeySegment
{
    short int nOffset;          /* Offset from beginning of structure in bytes */
    short int nSize;           /* Size of data item in bytes */
    int idDataType;            /* EVDT_MATH_NUMERIC or EVDT_STRING*/
} JDECMKEYSEGMENT;

struct _JDECMKeyStruct
{
    short int nNumSegments;
    JDECMKEYSEGMENT CacheKey[JDECM_MAX_NUM_SEGMENTS];
} JDECMINDEXSTRUCT;
```

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 single indexes. Declare the index structure as an array that holds more than one element for multiple indexes. You can create an unlimited number of indexes.
- Always use memset() for the index structure. When you use memset() for multiple indexes, multiply the size of the index structure by the total number of indexes.

- Always assign as elements the number of segments that correspond to the number of columns 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;
```

The following example illustrates a cache with multiple indices.

```

/*Declare the index array of 2 elements*/
JDECMSTRUCT Index[2];
int numIndexes=2;
/*Initialize the structure*/
memset(&Index,0x00,sizeof(JDECMSTRUCT)* numIndexes);
Index[0].nkeyID = 1;
Index[0].nNumSegments = 1;
Index[0].CacheKey[0].nOffset=offsetof(F0101,abat1);
Index[0].CacheKey[0].nSize=sizeof(f0110.abat1);
Index[0].CacheKey[0].idDataType=EVDT_STRING;
Index[1].nkeyID = 2;
Index[1].nNumSegments = 2;
Index[1].CacheKey[0].nOffset=offsetof(F0101,abac02);
Index[1].CacheKey[0].nSize=sizeof(f0110.abac02);
Index[1].CacheKey[0].idDataType=EVDT_STRING;
Index[1].nNumSegments = 1;
Index[1].CacheKey[1].nOffset=offsetof(F0101,abat1);
Index[1].CacheKey[1].nSize=sizeof(f0110.abat1);
Index[1].CacheKey[1].idDataType=EVDT_STRING;

```

Initializing the Cache

After you set up the index or indices, call `jdeCacheInit` or `jdeCacheInitMultipleIndex` to initialize (create) the cache.

Pass a unique cache name so that JDECACHE can identify the cache. Pass the index to this API so that the JDECACHE knows how to reference the data that will be stored in the cache. Because each cache must be associated with a user, you must also pass the user handle obtained from the call to `JDB_InitUser`. This API returns 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` and `jdeCacheInitMultipleIndex` call for that cache until it is terminated. The keys in the index must correspond in number, order, and type for that index each time that it is used.

After the cache has been initialized successfully, JDECACHE operations can take place using the JDECACHE APIs. The cache handle obtained from `jdeCacheInit` must be passed for 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:

- 1
- c
- i

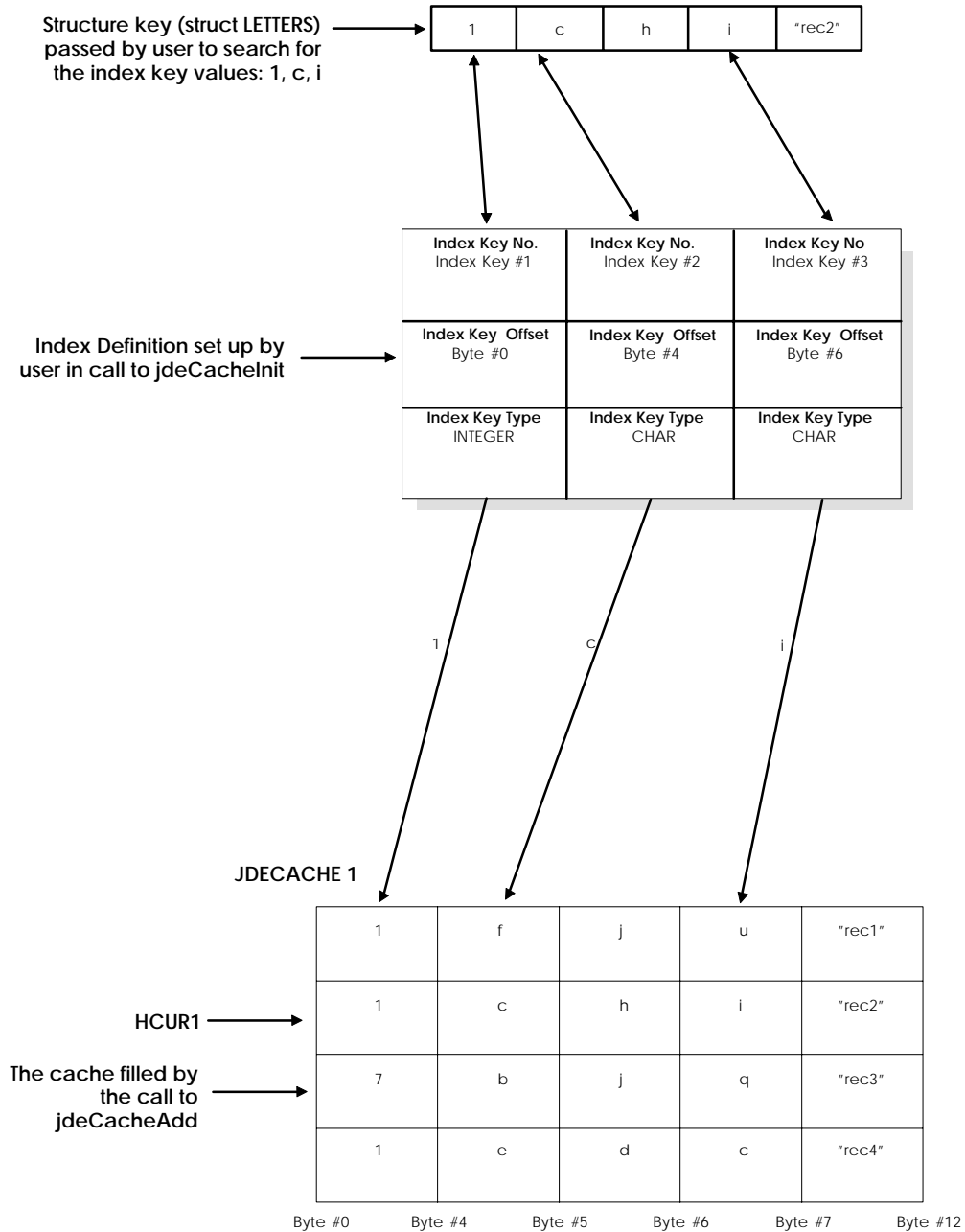
JDECACHE accesses the values that you pass in the structure at the byte offsets that were defined in the call to `jdeCacheInit`.

JDECACHE compares the values 1, c, 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 that were defined in the index passed to `jdeCacheInit`. The structure containing the keys when accessing a cache should be of 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.

The key for searches must always use the same structure type that is used to store cache records.

Using the `jdeCacheInit/jdeCacheTerminate` Rule

For every `jdeCacheInit` or `jdeCacheInitMultipleIndex`, there must be a corresponding `jdeCacheTerminate`, except if the same cache is used across business functions or forms. Then all unterminated `jdeCacheInit` or `jdeCacheInitMultipleIndex` calls must be terminated with a `jdeCacheTerminateAll`.

A `jdeCacheTerminate` call terminates the most recent corresponding `jdeCacheInit`. This means that the same cache can be used in nested business functions. In each function, perform a `jdeCacheInit` 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` or `jdeCacheInitMultipleIndex` calls or `jdeCacheTerminate` calls.

Using the Same Cache in Multiple Business Functions or Forms

If the same cache is required for two or more business functions or forms, call `jdeCacheInit` in the first business function or form, and add data to it. After exiting that business function or form, do not call `jdeCacheTerminate` because this removes your cache from memory. Instead, in the subsequent business functions or forms, call `jdeCacheInit` again with 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 where the cursor is currently pointing 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. See *JDECACHE Multiple Cursor Support* for more information.

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

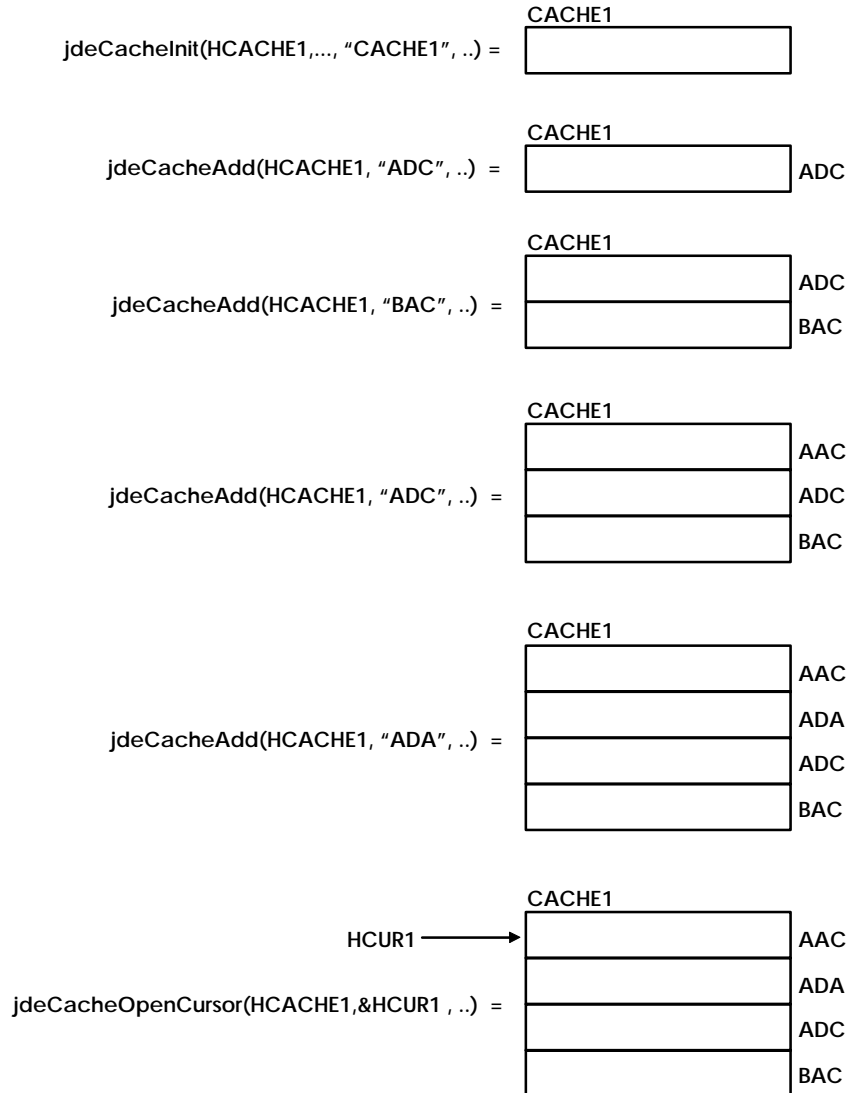
`HJDECURSOR` is the data type for the cursor handle. It must be passed to every JDECACHE Data Manipulation API except `jdeCacheAdd`.

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 advances 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 from which you want to start retrieving. This advances the cursor to the desired location in the dataset and retrieves the record. All subsequent calls to `jdeCacheFetch` will fetch records starting from the current cursor position in the data set until the end of the dataset, or until the program is stopped for another reason.

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 with a key that 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`.

If the index key is changed, cache resorts the records again, and the cursor points to the updated location. However, when you call `jdeCacheFetch`, the system retrieves the next record in the updated set. Consequently, the system might not retrieve the correct record because changing the index key caused the order of the records to change.

To update a sequential number of records, make a call to `jdeCacheFetchPosition` to get to the beginning of the sequence, if necessary. Then call `jdeCacheUpdate`, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`. This call updates just the record to which the cursor is pointing. To update the rest of the records in the sequence, call `jdeCacheFetch` repeatedly, passing the same `HJDECURSOR` that you used in the call to `jdeCacheFetchPosition`, until you get to the end of the sequence. A sequential update will not work correctly if you have changed any index key value. Sequential update works correctly if you are updating a non-index key value, however.

Deleting Records

If you want to delete a specific record with a known key, first call `jdeCacheFetchPosition` to point the cursor to the location of the record that matches the key. Next, call `jdeCacheDelete`, to remove the record from cache. Pass `jdeCacheDelete` the same `HJDECURSOR` that you used when you called `jdeCacheFetchPosition`. After deleting a record, use `jdeCacheFetch` to retrieve the record that followed the now deleted record. This process only works when you call `jdeCacheDelete`.

You can also delete a specific record by calling `jdeCacheDeleteAll` and passing it the full key with the specific record to be deleted. In this case, `jdeCacheFetch` will not work following `jdeCacheDeleteAll`, although you can work around this condition with `jdeCacheFetchPosition` or `jdeCacheResetCursor`.

To delete a sequential set of records, first call `jdeCacheFetchPosition` to point the cursor to the first record in the set or call `jdeCacheDeleteAll` to delete the first record in the set. Then, call `jdeCacheDelete` sequentially. In this case, `jdeCacheFetch` will not work following `jdeCacheDeleteAll`, although you can work around this condition with `jdeCacheFetchPosition` or `jdeCacheResetCursor`.

If you want to delete records that match a partial key, call `jdeCacheDeleteAll` and pass it a partial key. The system deletes all the records matching the partial key. After calling this API, `jdeCacheFetch` does not work.

The `jdeCacheFetchPosition` API

The `jdeCacheFetchPosition` API searches for a specific record in the dataset; thus, it requires a specific key. This API can perform full and partial key searches. See *JDECACHE Partial Keys* for a detailed explanation of partial keys. If you pass 0 for the number of keys, the system assumes you want to perform a full key search.

The jdeCacheFetchPositionByRef API

The jdeCacheFetchPositionByRef API returns the address of a data set. The API finds the one record in cache and returns a reference (pointer) to the data. jdeCacheFetchPositionByRef is designed to retrieve a single, large block of data that is stored in cache. If the cache is empty or has more than one record, this API fails.

Resetting the Cursor

JDECACHE Cursors supports multiple cursors as well as an unlimited number of cursor oscillations within the dataset. This means that the cursor can shuttle from beginning to end for an unlimited number of times. 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 to give 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 advised to close the cursor as soon as it is no longer needed to release the memory it requires.

JDECACHE Multiple Cursor Support

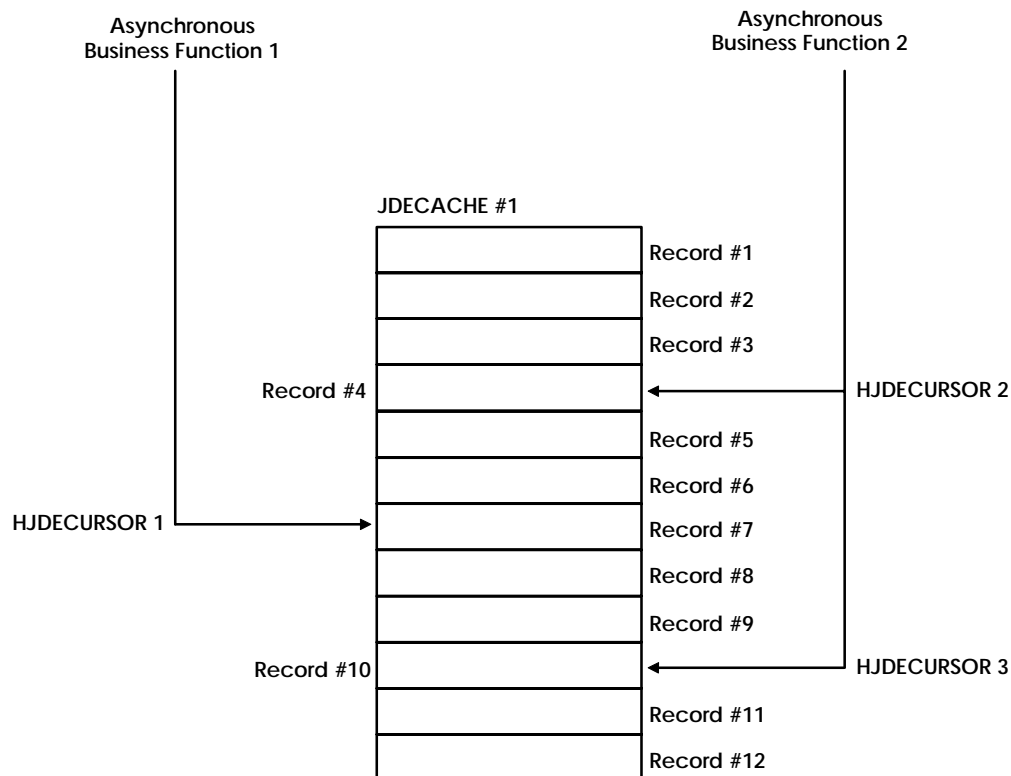
JDECACHE supports multiple open cursors. This means that each cache allows up to 100 open cursors to access it at the same time.

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 with relative positions within the cache that 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. Partial key components must appear in the the same order as the key components in the index (the index is 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 from to start fetching. 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 fourth 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 so that 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 an effective way of handling errors. All user errors handled by JDECACHE are communicated to the user through the JDE.LOG. The error statement in the log identifies the error and the cause.

The following are some of the errors you might find in the JDE.LOG:

```
510/441      Fri Apr 21 10:22:31 2000      jdecache401
             CAC0001077 - (jdeCacheInit) Initialization error.
             Re-initializing with incorrect segment number.
```

```
510/441      Fri Apr 21 10:22:31 2000      jdecache730
             CAC0001003 - (jdeCacheAdd) Empty data key encountered:
             jdeCacheAdd failed.
```

```
510/441      Fri Apr 21 10:22:31 2000      jdecache730
             CAC0001017 - (jdeCacheOpenCursor) Attempt to exceed total
             number of simultaneously open cursors (100) per cache failed.
```

JDEDEBUG.LOG indicates each JDECACHE API that was called and when it was called.

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
                    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->nKeyID = 1;
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 particular 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;
}

```



```

/*****
/*
/*                               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,numColsInIndex,&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 always 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*/

    /*****
    /*
    /*                               */
    /*                               */
    /*                               */
    /* 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 that 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 remain available.
- Before cache termination free memory of cache data structures, if necessary. Cache data structures are different from cache index structures. Terminating the cache frees cache index structures but does not affect 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. The number of keys used is passed to the function through the business function data structure standard data items.
- 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. The number of keys must remain constant.
- Compare value of defined # of keys in previous record with the next record to determine the 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.
*
*****/

```

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.

Additional Features



Processing Options

Processing options control how an interactive or batch application processes data. You can use processing options to change the way that an application or a report appears or behaves. You can attach unique processing options to different versions of the same application, which allows you to change the behavior of an application without creating a new application. You can use processing options to:

- Control the path that 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
- Specify the default version of an application or batch process

You can define processing options for an application that automatically display at runtime.

This section describes:

- Defining a processing options data structure (template)
- 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.

Caution: Changes to processing option text can conflict with processing option template changes. Template changes do not take effect until another package is built, but text changes 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 how you set up the application, one of the following events 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, or override location).

Each version of an application can be associated with a list of processing option values. The processing options that are specified at the time an interactive or batch application is launched are the ones that the application uses when it executes. This sequence prevents processing options from being changed between the time a batch application is submitted and the time it actually executes. Processing options may also be used for a specific execution of an application. These processing options are not permanently stored in the F983051 table, and they are only used for that specific execution.

Defining a Processing Options Data Structure (Template)

You can create a processing options data structure (template) that lists the values for data items passed to the application at runtime. Any changes that 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.

Defining a processing options data structure contains the following topics:

- Defining a template
- J. D. Edwards processing option naming standards
- Language considerations for processing options

Before You Begin

- Create a processing options data structure. See *Creating a Processing Options Data Structure*.

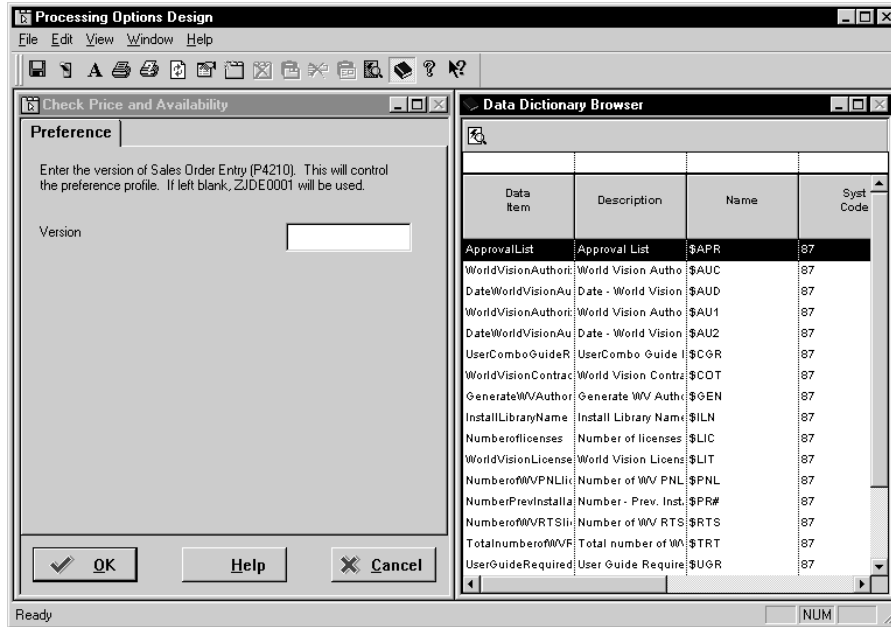
Defining a Template

Use the following process to determine how a processing options data structure looks and behaves. The structure itself must already exist. See *Creating a Processing Options Data Structure* for detailed instructions on creating the template object.

► **To define a processing options data structure (template)**

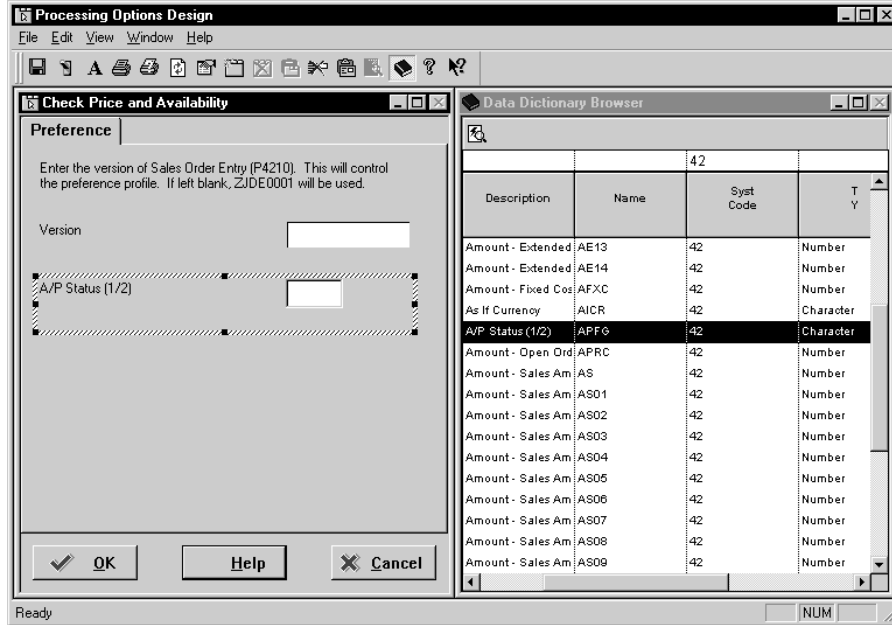
1. On Object Management Workbench, check out the processing options data structure with which you want to work.
2. Ensure that the data structure is highlighted, and then click the Design button in the center column.
3. On Processing Option Design, click the Design Tools tab, and then click *Start the Processing Option Design Aid*.

The Processing Options Design tool launches. The area on the left of the form displays how the processing option will look to the user.

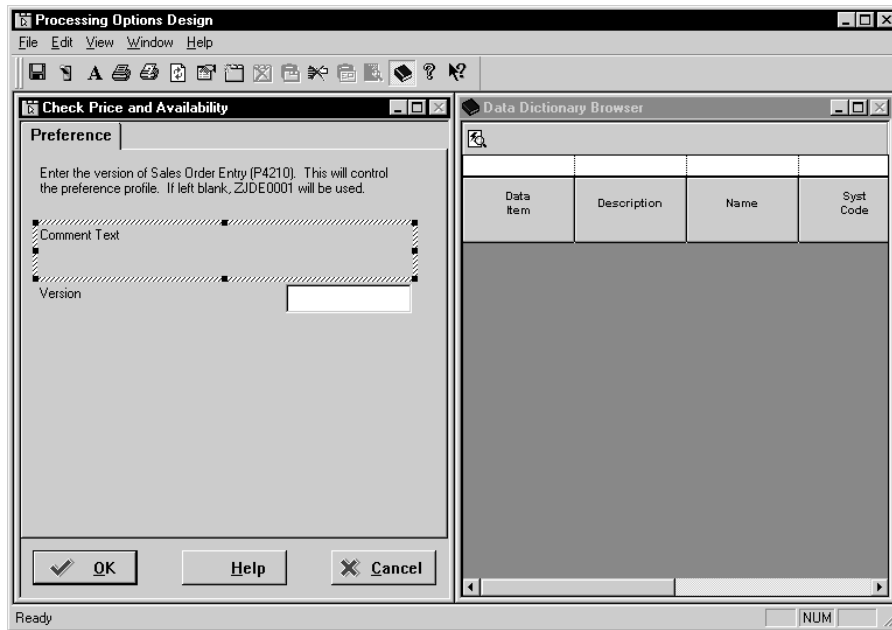


4. Use one of the following methods to choose the data items that you wish to add to your processing options:
 - Double-click on the item in the Data Dictionary Browser, and the item will appear in the left side of the form under the tab.
 - Drag the item from the Data Dictionary Browser to the position where you want it in the structure members.
5. Click an item to edit it.
 - Use the hatching around the control to reposition it.
 - Select text, and delete or overwrite it.

Processing Options Design automatically adjusts the size and position of data items to fit the width of the tab.



- Choose the text button (A) to add comments.



- Choose an object in the area on the left side of the form, and choose Properties from the View menu.

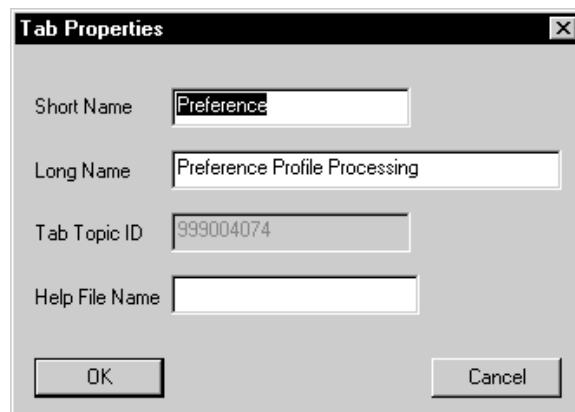


If you are on a data item, you can view its properties and change the Item Name if necessary. The Item Name should be unique.

You can click the Help Override Data Item tab to add an alternate szDict from which to get the help.

8. 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.



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.

9. 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 <code>TBLE</code> , or business functions is <code>BSFN</code> .
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.

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: <ul style="list-style-type: none"> • 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.

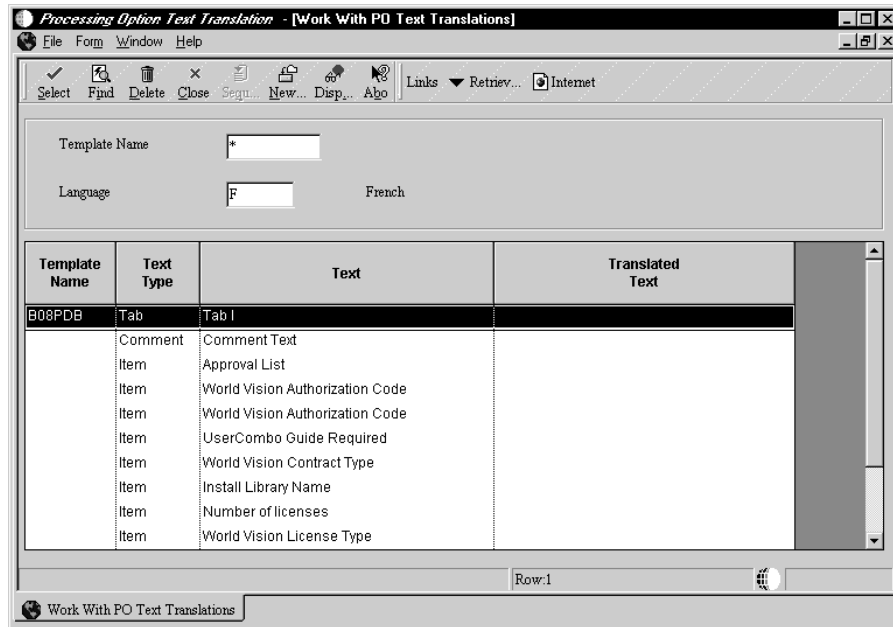
Language Considerations for Processing Options

You can change a processing options template to incorporate language features. Language considerations for processing options contains the following tasks:

- Changing a template for text translation
- Adding a template with translated text

▶ **To change a template for text translation**

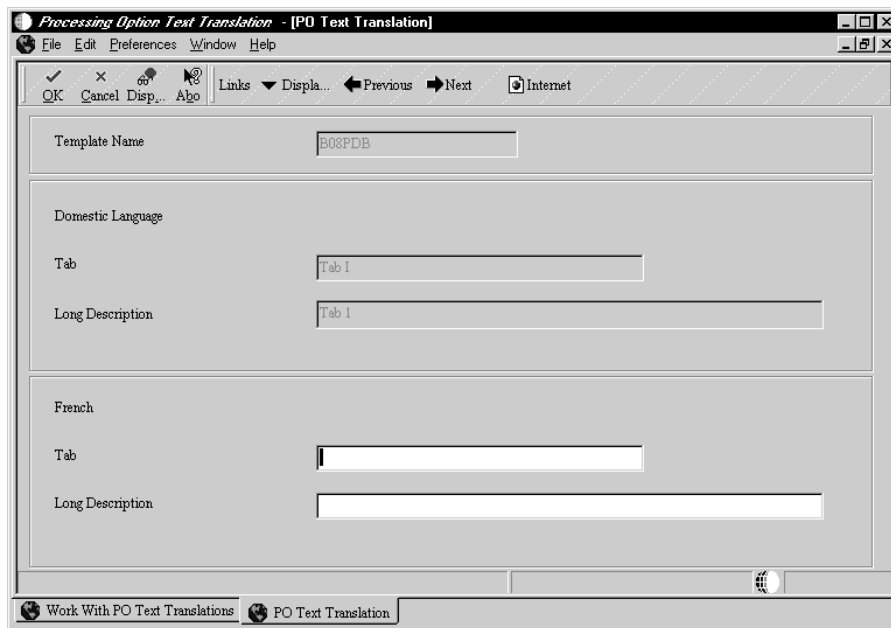
1. From System Administration Tools (GH9011), choose Processing Options Text Translation.
2. Choose the processing option template that you wish to change.



Work with PO Text Translations displays processing option text for the processing option template and language that you specify in the filter fields.

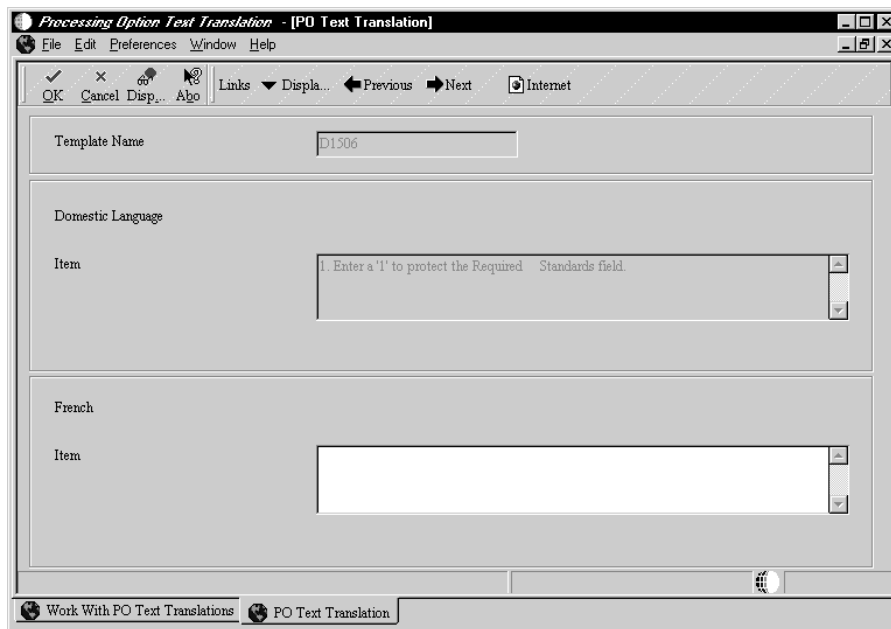
3. Select the item that you wish to change and enter the new text.

You can change a tab.



You can change an item.

You can change a comment.



▶ **To add a template with translated text**

When you add a new processing option template for an application that is language enabled, complete the following tasks:

1. Create the application.
2. Create the Processing Template for the base language.
3. Add the language text (refer to the instructions above).

Attaching a Processing Options Template

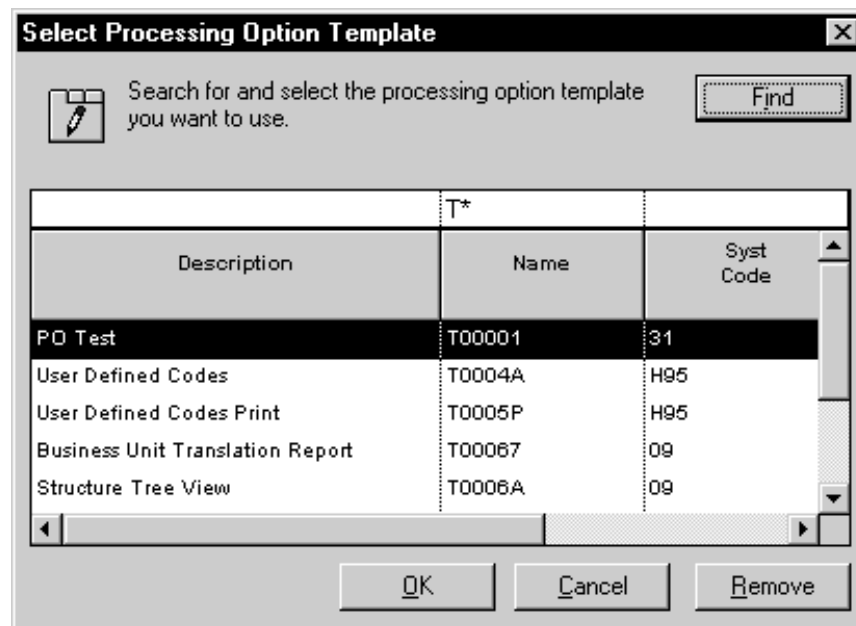
You must attach a processing options template (data structure) to an application to enable processing options at runtime. You can use processing options to set up default values, control formats, control report breaks, control totaling, and control how a report processes data. A processing option template:

- 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

1. On Application Design Aid, 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.

Caution: 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, first remove all existing versions of the application. 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 of them are required for the complete task.

Transaction Processing ensures that related data is added to or deleted from the database simultaneously, thus preserving data integrity in your application. In transaction processing, data is not written to the database until a commit command is issued. When this happens, data is permanently written to the database.

For example, if a transaction comprises database operations to update two database tables, either all updates will be made to both tables, or no updates will be made to either table. This condition guarantees that the data remains in a consistent state and the integrity of the data is maintained.

You see a consistent view of the database during a transaction. You do not see changes from other users during a transaction.

Transaction Processing ensures that transactions are:

- Atomic - Either all database changes for an entire transaction are completed or none of the changes 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 event makes the successful end of a transaction.

Two-Phase Commit (Manual Commit Mode)

A two-phase commit coordinates 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 event makes 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
- Setting the jde.ini for transaction processing and lock manager

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. Journaling can increase performance overhead due to the additional processing required. Contact your DB2 administrator if you have problems with this process.

General messages and errors for transaction processing are written to the jde.log or jdedebug.log.

Data Interdependence

Data interdependence refers to the data elements that make a transaction complete. For example, a voucher has records in both the 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	Form	TP On	TP Off	Form, BSFN 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. Form1 operates in Auto Commit mode and Form2 operates in Manual Commit mode.
	Form2	X			
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 (child) is in Manual Commit mode and the interconnect is ignored.
	Form2	X			
G	Form1	X			Transaction processing is On for both forms. Because the Include in Parent flag is Off, each form is a transaction boundary and a commit is issued for each.
	Form2	X			
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. The transaction boundary encompasses both forms. Form2 is a child of Form1.
	Form2	X			

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.

Note: When you use business functions within a transaction, you must be careful not to cause a deadlock. If you split the logic for manipulating a table between two functions, you 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. The servers and client must all commit the transaction, or the transaction rolls back on the servers and client.

Transaction Processing System Functions

Several transaction processing system functions are available. You might need to use system functions for additional transaction processing functionality.

For example, 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 that 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 report

Defining Transaction Processing for a Form

To define transaction processing for a form, you must specify the Transaction option on Form Properties in Form Design. This requirement means that all data for the form is committed to the database at the same time.

If a transaction includes a single form, then this is the only action that is required because the form itself is the transaction boundary.

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

1. On Forms Design, 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. Enable the Transaction Processing flag through a form interconnection in Event Rules Design to extend the boundary.

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

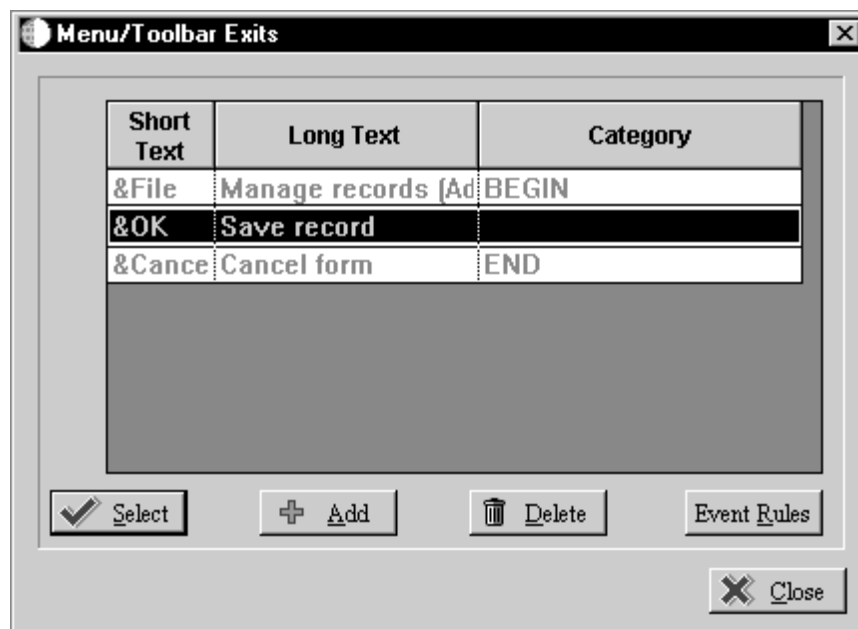
- In Form Design, define form properties for each form within the transaction boundary to include transaction processing.

Extending a Transaction Boundary between Forms

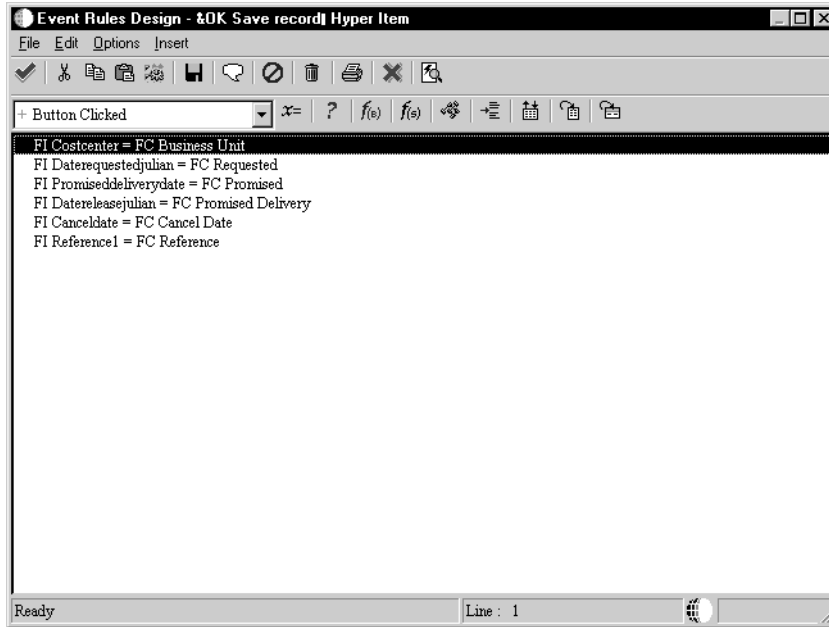
If the parent form uses manual commit, the form that you connect it to will also use manual commit.

▶ To extend the transaction boundary between forms

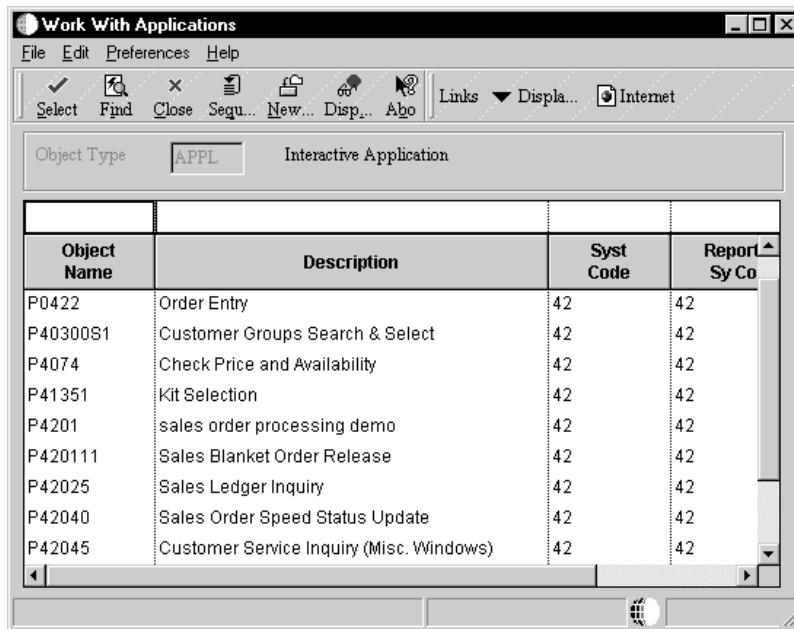
1. On Forms Design, on the parent form with which you are working, from the Form menu, choose Menu/Toolbar Exits.



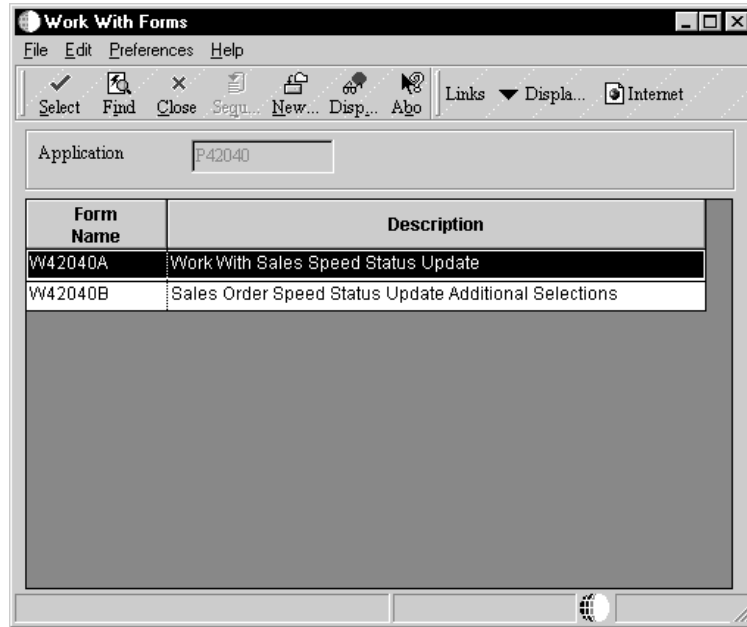
2. Choose the OK row and click the Event Rules button.



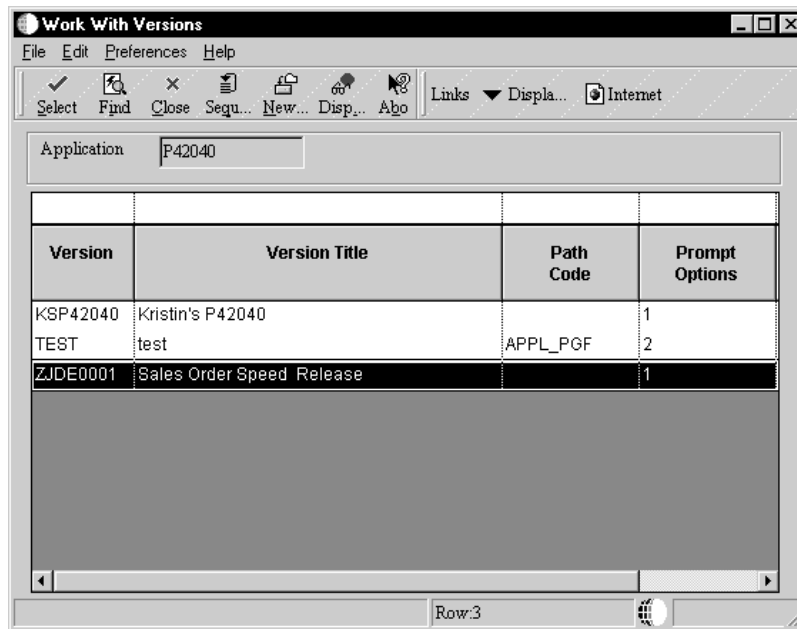
- From Event Rules Design, choose the *Button Clicked* event, and click the Form Interconnect button.



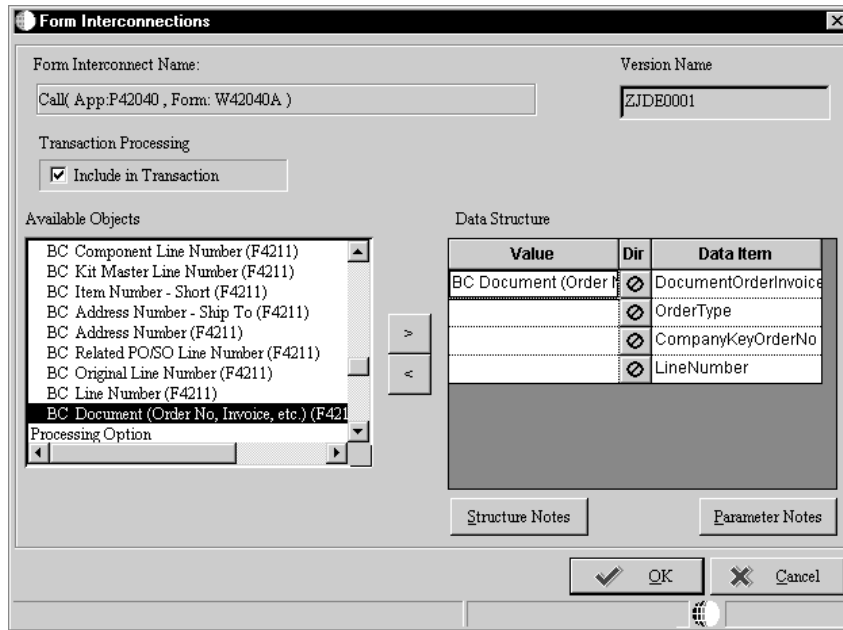
- On Work with Applications, choose the application that you wish to use.



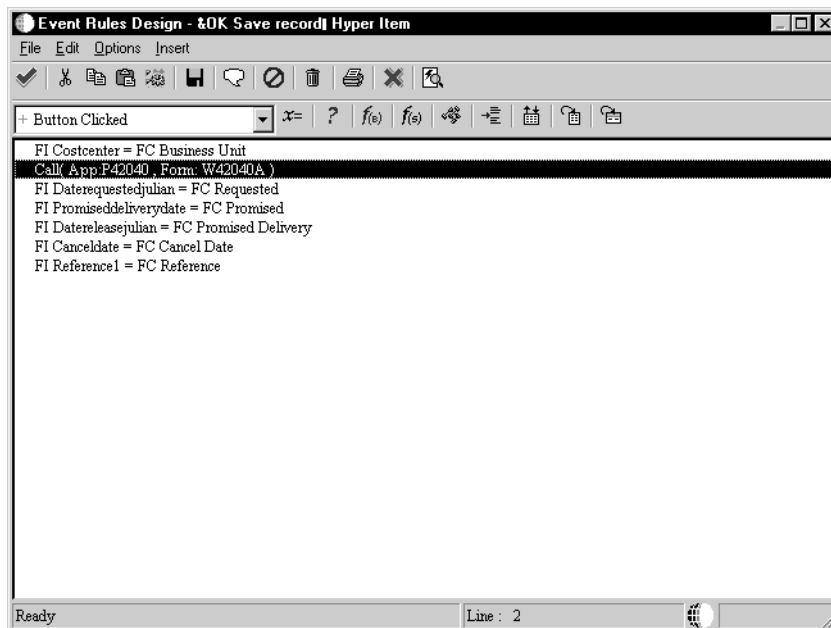
- On Work with Forms, choose the form that you wish to include in the transaction boundary.



- On Work with Versions, choose the version of the application you wish to use.



7. On Form Interconnect, click the following Transaction Processing option and click OK:
 - Include in Transaction



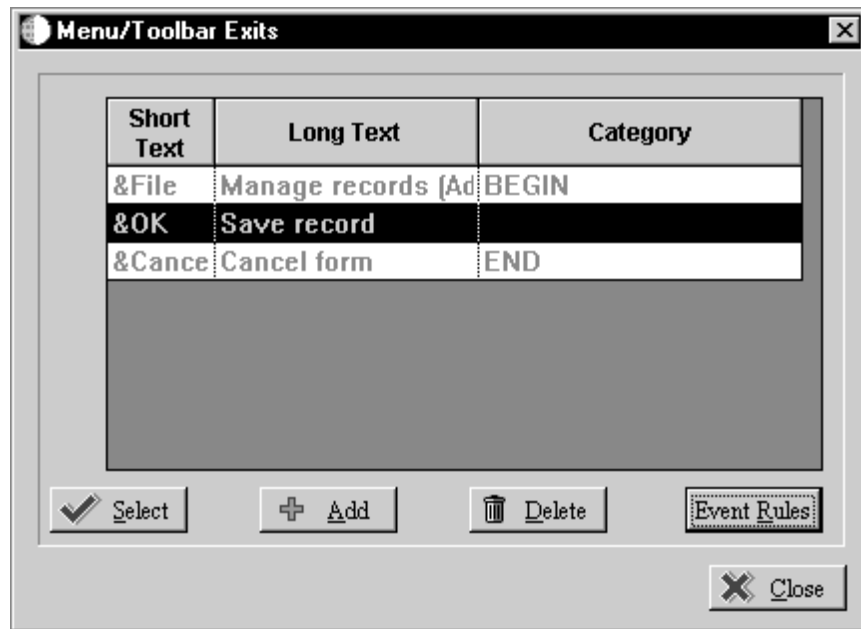
Extending a Transaction Boundary Using Business Functions

You can include a business function in a transaction boundary. If the parent form uses auto-commit, the business function that you extend the transaction boundary to will also use auto-commit. Any business function not marked as

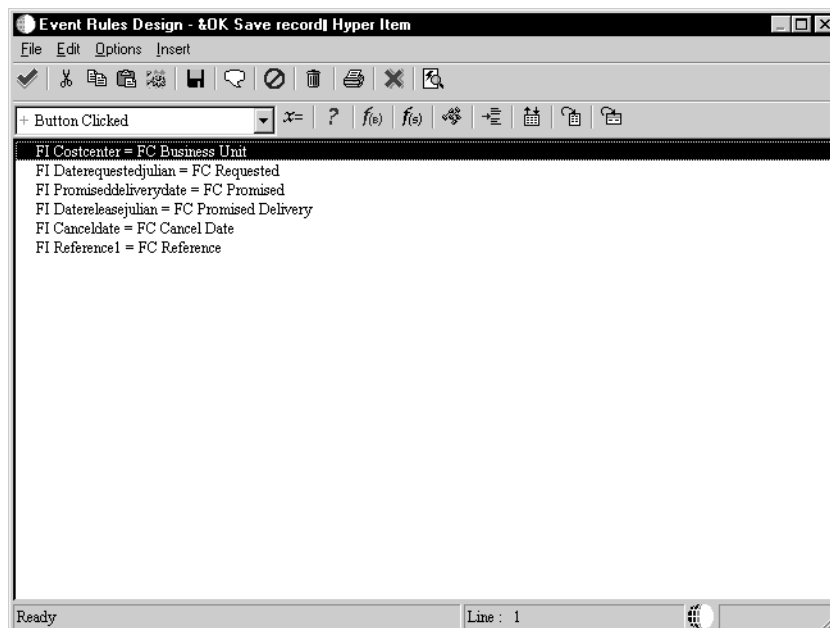
“Include in Transaction” will use auto-commit. Business functions that process asynchronously can also be included in a transaction.

► **To extend a transaction boundary using business functions**

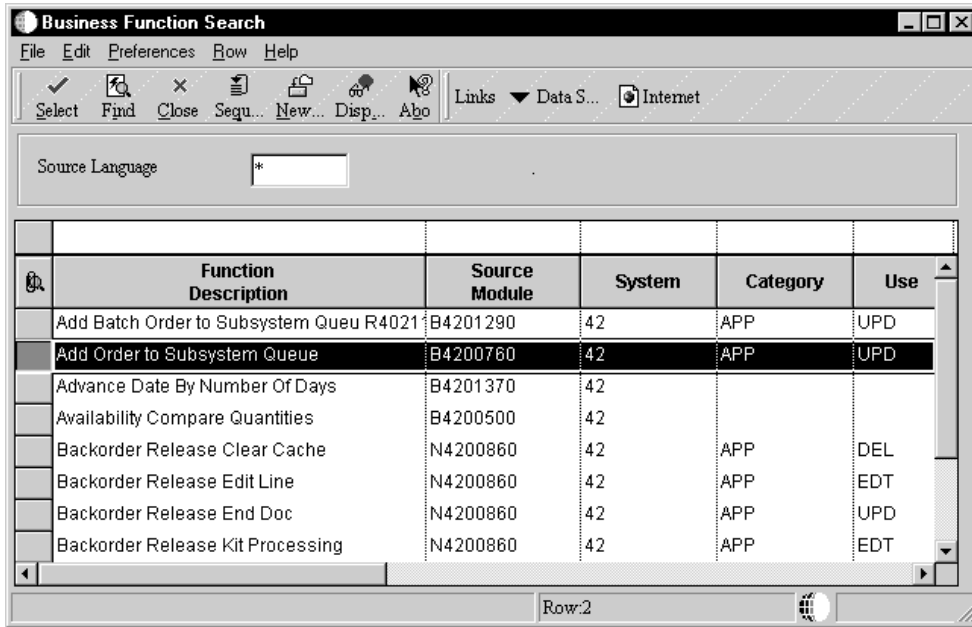
1. On Forms Design, on the parent form with which you are working, from the Form menu, choose Menu/Toolbar Exits.



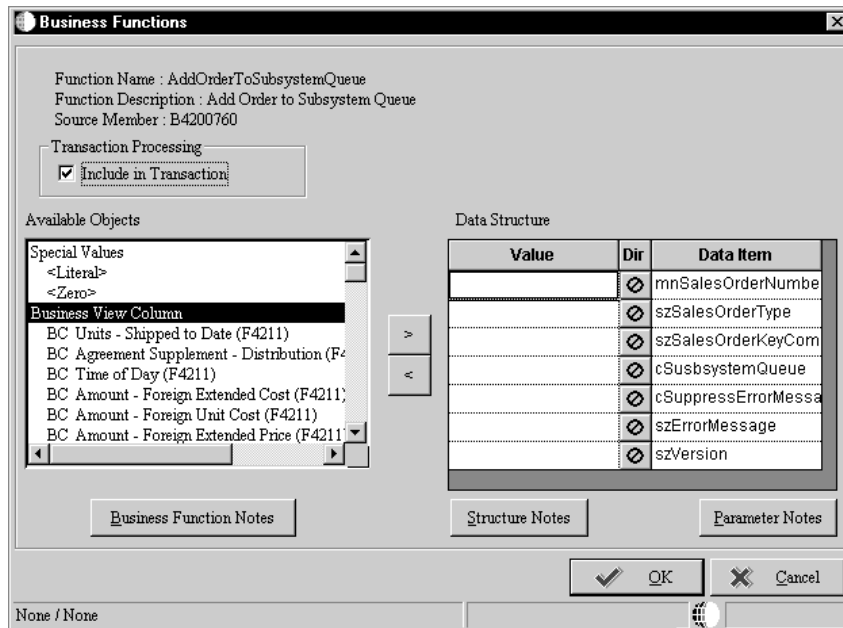
2. Choose the OK row and click the Event Rules button.



- From Event Rules Design, choose the *Button Clicked* event, and click the Business Functions button.

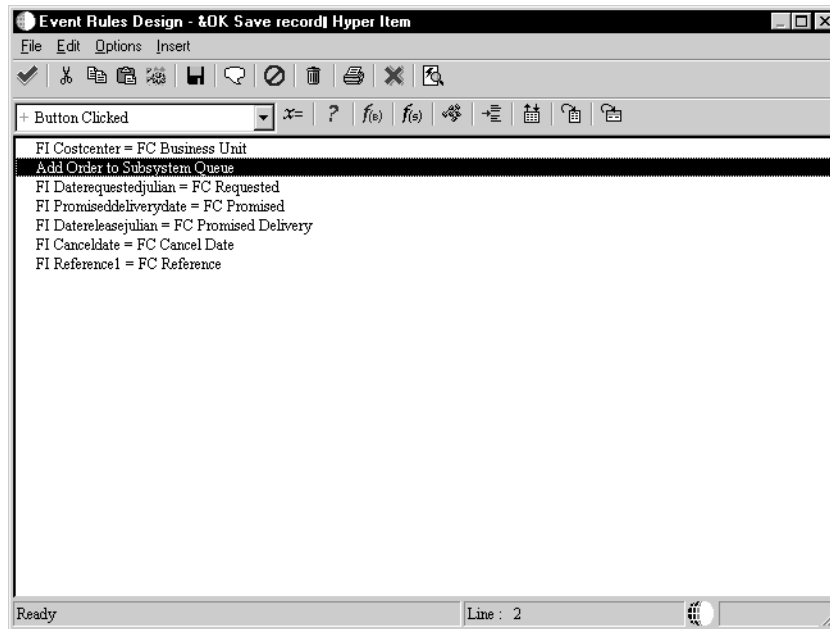


- From Business Function Search, choose the business function that you wish to include in the transaction boundary.



- On Business Function, click the following Transaction Processing option:
 - Include in Transaction

Business Functions marked for both “Asynchronous” and “Include in Transaction” are included in the transaction boundary.

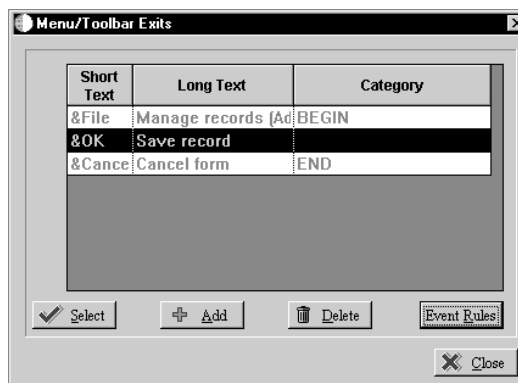


Extending a Transaction Boundary Using Table I/O

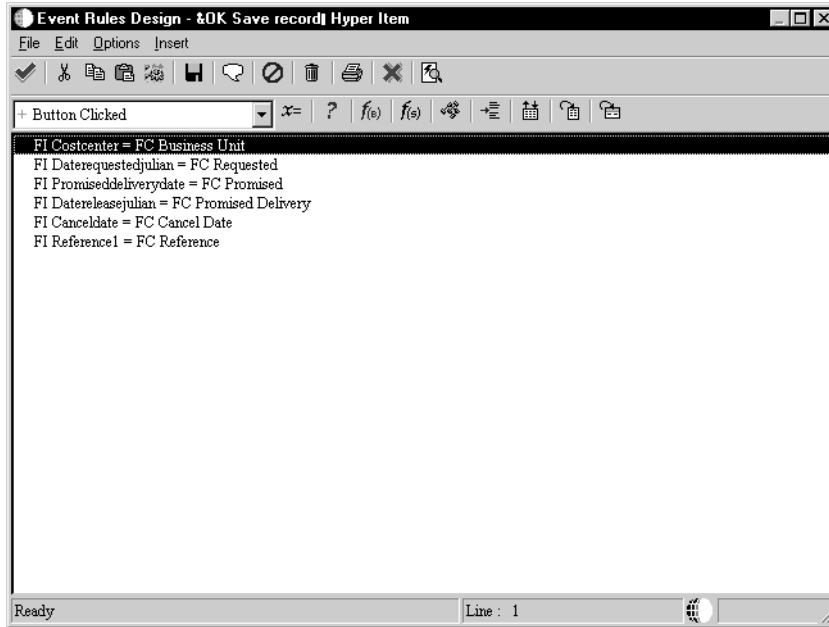
Transaction processing is only available for the Open Table operation in Table I/O. The opening of a table determines if interaction with that table will be manual commit (part of a transaction) or auto-commit. Any Open Table operation not marked as “Include in Transaction” will use auto-commit.

► To extend a transaction boundary using table I/O

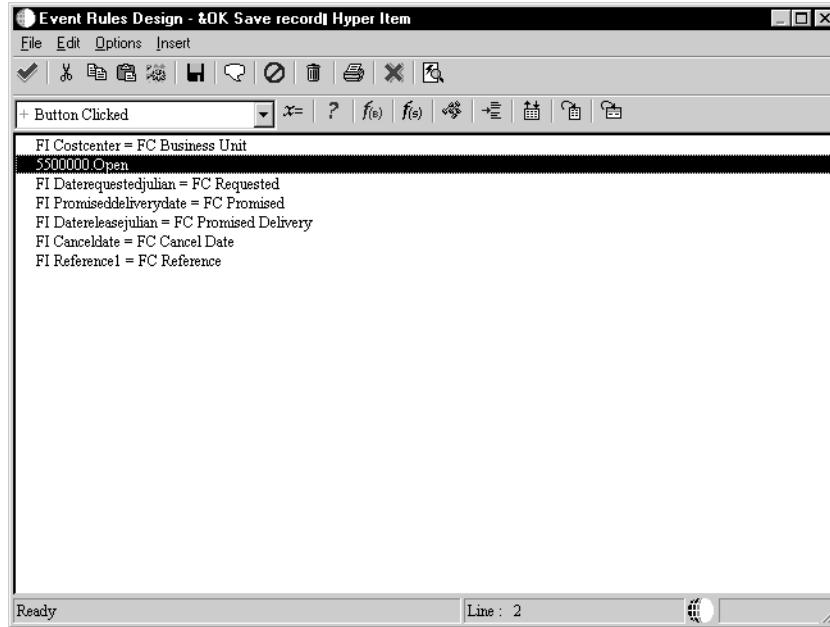
1. On Forms Design, on the parent form with which you are working, from the Form menu, choose Menu/Toolbar Exits.



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.
4. On Insert TableIO Operation, choose the Open option under Advanced Operations, and then click Next.
5. On Data Source, click Advanced Options.
6. On Advanced Options, choose the *Include in Transaction* option, and then click OK.
7. On Data Source, 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, click the Advanced tab for report properties and choose Transaction Processing.



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.

Setting the jde.ini for Transaction Processing and Lock Manager

You must modify the enterprise server and workstation jde.ini files to enable transaction processing.

For each OneWorld workstation, you must enable transaction processing by changing settings in the workstation jde.ini file. You should make these changes on the deployment server to the resident jde.ini file that is delivered to workstations through package deployment, and then deploy a package with the changed jde.ini file.

This chapter contains the following topics:

- Understanding concurrent release support
- Understanding transaction processing and lock manager logging
- Configuring the jde.ini for transaction processing

Understanding Concurrent Release Support

Prior to release B73.3, transaction processing was primarily controlled by settings in the [TP MONITOR ENVIRONMENT] section of the jde.ini. On the server, this section contains nine settings. On the workstation, this section contains seven settings.

For release B73.3, the [TP MONITOR ENVIRONMENT] section has been removed from the jde.ini and replaced with the [LOCK MANAGER] section. This new section contains three settings in both the server and client jde.ini files.

Settings that used to be in the [TP MONITOR ENVIRONMENT] section were removed because they were either obsolete or assigned an internal default value.

The following list provides the reasons the settings in the [TP MONITOR ENVIRONMENT] section were eliminated for release B73.3:

Setting	Reason for eliminating
Status	Obsolete. As of release B732.2, the TP monitor is always set to ON.
LogPath	Assigned the base directory from the jde.ini file.
LogStatements	Obsolete. Statements are always logged.
LogBufferSize	An internal default (1 MB) is used.
DisplayServerErrorMsg	Obsolete. The client always displays server error messages.
ServerRetryInterval	An internal default interval is used.
RegistryCleanupInterval	An internal default interval is used.
RegistryRecordLifeSpan	An internal default span is used.
ServerTimeout	This value is provided by JDENET settings.

During a transition period, both sections will be supported concurrently. There are three possible scenarios during this transition period:

- **The [LOCK MANAGER] section does not exist.** In this scenario, OneWorld checks for settings in the [TP MONITOR ENVIRONMENT] section.
- **Both [LOCK MANAGER] and [TP MONITOR ENVIRONMENT] sections exist.** In this scenario, OneWorld uses the settings in the [LOCK MANAGER] section.
- **Neither section exists.** In this scenario, transaction processing cannot be started and a failure occurs.

Understanding Transaction Processing Logging

The commit coordinator acts in two phases. In the first phase, it instructs the Log Manager to flush the logs for each data source to hard disk for a permanent backup storage. The logs contain every database operation that was carried out.

The first phase ensures that if any of the data sources fail to commit after the others have committed, all databases can be returned to a consistent state by referring to the contents of the logs. If all the logs for each of the data sources are flushed successfully, then the second phase begins.

In the second phase, the coordinator instructs each of the data sources to commit its respective transaction. If any of the data sources fails to commit, a commit log report is generated from the logs that were generated in phase one (this is written to the directory specified in the LOGPATH in the jde.ini which contains a listing by data source of all the SQL statements that were part of the transaction). The commit log also contains the details as to which data sources passed and which ones did not.

The log can help the database administrator to manually synchronize the data sources so that they are all in a consistent state. The commit log report is only generated when at least one data source fails to commit. If all data sources successfully commit, then no commit log report is generated, and all the logs from phase one are deleted by the Log Manager.

Since B73.3, the log file is placed in the directory specified in the jde.ini [INSTALL] section.

Setting the jde.ini for Transaction Processing and Lock Manager

As previously explained, during a transitional period, both the [TP MONITOR ENVIRONMENT] and [LOCK MANAGER] sections are supported.

If you are using a release of OneWorld prior to B73.3, enter settings for the [TP MONITOR ENVIRONMENT] section. If you are using release B73.3 or higher, enter settings for the [LOCK MANAGER] section.

Note: The following settings are applicable to the Lock Manager:

- Server
- RequestedService
- AvailableService

The remainder of the settings relate to transaction processing.

Settings for the [TP MONITOR ENVIRONMENT] Section (Prior to B73.3)

The following tasks describe how to enter settings for the [TP MONITOR ENVIRONMENT] section of the jde.ini file, both for the server and for the workstation. These settings will be used only if you are using a OneWorld release prior to B73.3.

► **To enter [TP MONITOR ENVIRONMENT] settings for the server**

1. Locate the enterprise server's jde.ini file.
2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure accuracy of the following settings:

```
[TP MONITOR ENVIRONMENT]
Status=status value
LogPath=log path
LogStatements=log on/off value
LogBufferSize=log buffer value
RequestedService=service value
Server=server name
ServerTimeout=timeout value
AvailableService=service value
RegistryCleanupInterval=cleanup value
RegistryRecordLifeSpan=life span value
RegistryRecordLifeSpan=lifespan value
LogServices=service value
```

See the following table for explanations of the variables above.

Setting	Value
Status	Indicates whether transaction processing is on or off. The transaction processing monitor should usually be on unless you are not using transaction processing in any of your applications, or if you want to temporarily disable transaction processing, such as for testing. Valid values are ON and OFF. As of B73.2, transaction processing cannot be turned off, so this value is ignored in versions B73.2 and higher.
LogPath	<p>This setting specifies the directory in which the transaction logs are placed. This path should correspond to the location of your jde.log and jdedebug.log.</p> <p>For example, on a UNIX machine, the path might be:</p> <pre>/u10/owdevel/tc283984/b73.2</pre> <p>For more information about logging, see <i>Understanding Transaction Processing Logging</i>.</p>
LogStatements	This setting specifies whether the transaction monitor should keep a log of every operation performed within a transaction. Valid values are ON and OFF.

Setting	Value
LogBufferSize	This setting indicates the number of bytes set aside to hold the operations being logged in memory before they are copied to disk. This value is a OneWorld internal default value and should not be changed.
RequestedService	This setting specifies the service that the client requests from the server. Valid values are: <ul style="list-style-type: none">• TS Time stamp service is requested• NONE No service is requested
Server	Specifies the server that is hosting the TMS. For example, a server name might be intelnta.
ServerTimeout	The timeout in seconds for all the network operations. This value can be adjusted based on the network traffic. It is necessary for the workstation jde.ini file. It is also necessary for the server jde.ini file in case a batch job is running on the server. This value is a OneWorld internal default value and should not be changed.
AvailableService	Indicates the service that this Transaction Management Server is offering. When the Transaction Manager on the workstation is initialized, it queries the TMS for this value. This is called TM-TMS handshaking. If this value is the same as the one that the workstation has in its jde.ini file, then that service will be invoked at the appropriate times when OneWorld is running. Valid values are: <ul style="list-style-type: none">• TS Record Change Detector (Timestamp Service)• NONE No service is available
RegistryCleanup Interval	The period after which all the expired records are deleted from the TMS record registry. This interval is specified in minutes. This value is a OneWorld internal default value and should not be changed.
RegistryRecordLife Span	The maximum period during which a record can exist in the TMS record registry. After this period the record will expire and will be deleted. This life span is specified in minutes. This value is a OneWorld internal default value and should not be changed.

Setting	Value
LogServices	<p>This setting turns on the Trace Log for TMS, and supplements the Jde.log file. Valid values are:</p> <ul style="list-style-type: none">• 1 Tracing for TMS is On• 0 Tracing for TMS is OFF <p>The default value for this setting is 0. You should turn tracing for TMS on only after all other debugging methods have been exhausted.</p>

► **To enter [TP MONITOR ENVIRONMENT] settings for the workstation**

Note: Be sure to enable transaction processing on the server before enabling it on the workstation. If you try to set up the workstation jde.ini file before you have set up the server jde.ini, you could be requesting a service on the server that is not yet available, which will generate an error.

1. Locate the jde.ini file that will be sent to the workstation as part of a package installation. This file is located on the OneWorld deployment server in the release share path:

```
\\Bxxx\CLIENT\MISC\jde.ini
```

where xxx is the installed release level of OneWorld. For example, B732.

2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure accuracy of the following settings:

```
[TP MONITOR ENVIRONMENT]
Status=status value
LogPath=log path
LogStatements=log on/off value
LogBufferSize=log buffer value
RequestedService=service value
Server=server name
ServerTimeout=timeout value
```

See the following table for explanations of the variables above.

Setting	Value
Status	Indicates whether transaction processing is on or off. The transaction processing monitor should usually be on unless you are not using transaction processing in any of your applications, or if you want to temporarily disable transaction processing, such as for testing. Valid values are ON and OFF. As of B73.2, transaction processing cannot be turned off, so this value is ignored in versions B73.2 and higher.
LogPath	<p>This setting specifies directory in which the transaction logs are placed. This path should correspond to the location of your jde.log and jdedebug.log.</p> <p>For example, on a UNIX machine, the path might be:</p> <pre>/u10/owdevel/tc283984/b73.2</pre> <p>For more information about logging, see <i>Understanding Transaction Processing Logging</i>.</p>
LogStatements	This setting specifies whether the transaction monitor should keep a log of every operation performed within a transaction. Valid values are ON and OFF.
LogBufferSize	This setting indicates the number of bytes set aside to hold the operations being logged in memory before they are copied to disk. This value is a OneWorld internal default value and should not be changed.
RequestedService	<p>This setting specifies the service that the client requests from the server. Valid values are:</p> <ul style="list-style-type: none">• TS Time stamp service is requested• NONE No service is requested
Server	Specifies the server that is hosting the TMS. For example, a server name might be intelnta.
ServerTimeout	The timeout in seconds for all the network operations. This value can be adjusted based on the network traffic. It is necessary for the workstation jde.ini file. It is also necessary for the server jde.ini file in case a batch job is running on the server. This value is a OneWorld internal default value and should not be changed.

The last three lines of the section pertain to record change detection and must be set if you want the workstation to perform “record is changed” database locking (as explained under the *Record is Changed* topic).

Note: Instead of deploying a package, you could also manually copy the jde.ini file to all workstations.

Settings for the [LOCK MANAGER] Section (B73.3 and higher)

The following tasks describe how to enter settings for the [LOCK MANAGER] section of the jde.ini file, both for the server and for the workstation. These settings will be used even if you entered values for the [TP MONITOR ENVIRONMENT] section.

► To enter [LOCK MANAGER] settings for the server

1. Locate your enterprise server jde.ini file.
2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure accuracy of the following settings:

```
[LOCK MANAGER]
Server=server name
AvailableService=available server service
RequestedService=client service request
```

See the following table for explanations of the variables.

Setting	Value
Server	<p>This setting indicates the lock manager server to be used to process records. The value for this setting is the name of the server acting as the lock manager. For example, a server name might be <code>intelnta</code>.</p> <p>If the client is used as a server, such as in cases where batch applications are running on the workstation, this setting must match the same entry in the workstation jde.ini file's [LOCK MANAGER] section.</p>
AvailableService	<p>This setting indicates the available service of the server. Valid values are:</p> <ul style="list-style-type: none">• TS Time stamp service is available• NONE No service is available <p>This setting applies only to servers.</p>

Setting	Value
RequestedService	This setting indicates the type of service that the client requests from the server. Valid values are: <ul style="list-style-type: none">• TS Time stamp service is requested• NONE No service is requested

Caution: Be sure to enable transaction processing on the server before enabling it on the workstation. If you try to set up the workstation jde.ini file before you have set up the server jde.ini, you could be requesting a service on the server that is not yet available, which will generate an error.

▶ To enter [LOCK MANAGER] settings for the workstation

1. Locate your workstation jde.ini file.
2. Using an ASCII editor such as Notepad, view the jde.ini file to ensure accuracy of the following settings:

```
[LOCK MANAGER]
Server=server name
RequestedService=client service request
```

See the following table for explanations of the variables.

Setting	Value
Server	This setting indicates the lock manager server to be used to process records. The value for this setting is the name of the server acting as the lock manager. For example, a server name might be intelnta. If the client is used as a server, such as in cases where batch applications are running on the workstation, this setting must match the same entry in the workstation jde.ini file's [LOCK MANAGER] section.
RequestedService	This setting indicates the type of service that the client requests from the server. Valid values are: <ul style="list-style-type: none">• TS Time stamp service is requested• NONE No service is requested



Record Locking

OneWorld does not implement any data-locking techniques. It relies on the native locking strategy of the vendor database management system. This process 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.

This section includes:

- Understanding record locking



Understanding Record Locking

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 he or she wants 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. This type of database locking prevents a user from updating a record that changes during the time the user is inquiring about it. If the record has changed, the user must select the record again and then make the change. This 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.
10:05	User B selects Address Book record “1001” to inspect it. Both users now have Address Book record “1001” open.

Time	Action
10:10	<p>User B updates a field in Address Book record “1001” and clicks OK.</p> <p>OneWorld updates Address Book record “1001” with the update that User B made.</p>
10:15	<p>User A updates a field in Address Book record “1001” and clicks OK.</p> <p>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 that User A was viewing it. For User A to change the record, User A must re-select it and perform the update.</p>

When “record change detection occurs”, OneWorld displays a message indicating that the record has been changed since it was retrieved.

See Also

- Setting jde.ini Files for Transaction Processing* in the *OneWorld System Administration* guide for more information about enabling record change detection.

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, verifies 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 closely to the update as possible.

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. Some fundamental requirements for an enterprise operating internationally include:

- Conversion of foreign currencies to the local currency
- Conversion of the different local currencies into one currency for reporting and comparisons
- Adhering to regulations defined in the countries of operation
- Revaluation of currencies due to 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, instead of the system, to control currency, provides greater flexibility and easier maintenance. Some of the advantages in allowing you to control currency are:

- Additional currency tables do not require changes to system modules. Only new business functions need to be added.
- Business logic is captured in business functions, rather than in a system module 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.

Working with currency includes the following topics:

- Understanding the build triggers option
- Understanding how table event rules work with currency processing
- Setting up currency conversion
- Showing currency-sensitive controls
- Creating a currency conversion trigger

Understanding the Build Triggers Option

The Build Triggers option performs the following steps:

- Converts event rules to C source code.
 - This creates the files *OBNM.c* and *OBNM.hxx* (where *OBNM* is the Object Name). The source file will contain one function per TER event.
 - For example, if you are working with table F0411, the Build Triggers option will create a C source member called F0411.c. You can browse through the C code and ensure that all the parameters are set up correctly. An error log will be generated if anything goes wrong during the ER-to-C conversion. The error table is called eF0411.log.
- Compiles the new functions and adds them into JDBTRIG.DLL (this is the Consolidated DLL that contains TER functions).

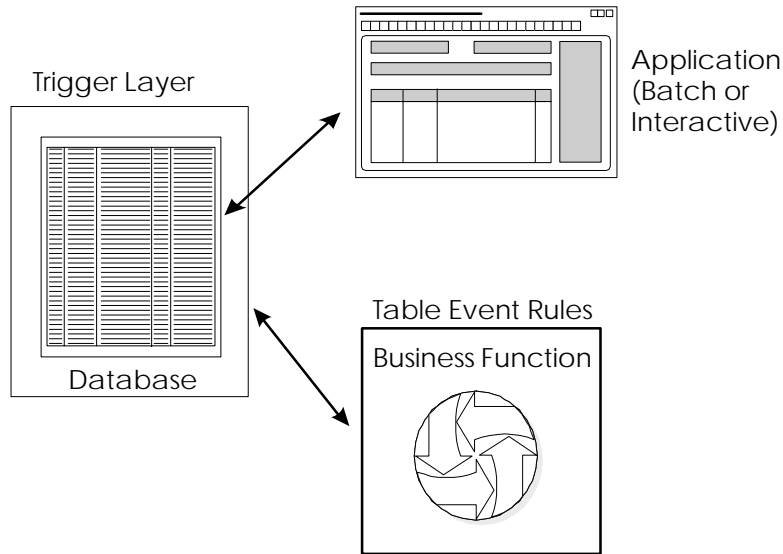
Understanding How Table Event Rules Work with Currency Processing

The *Currency Conversion* event runs if currency processing is 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

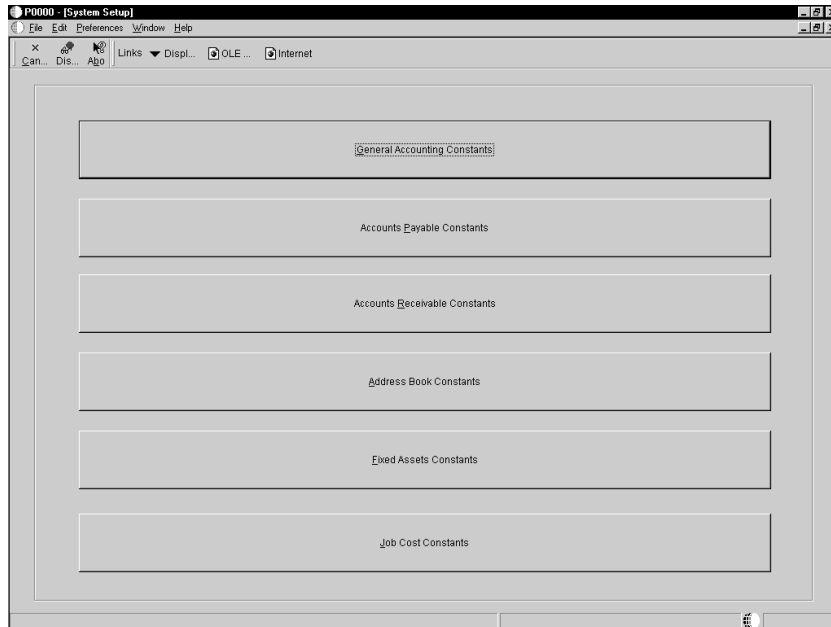
When passing Math_Numeric currency fields into a business function, the currency values in the respective data structure must be populated. Math_Numeric work fields that contain currency values also need the proper currency information.

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.

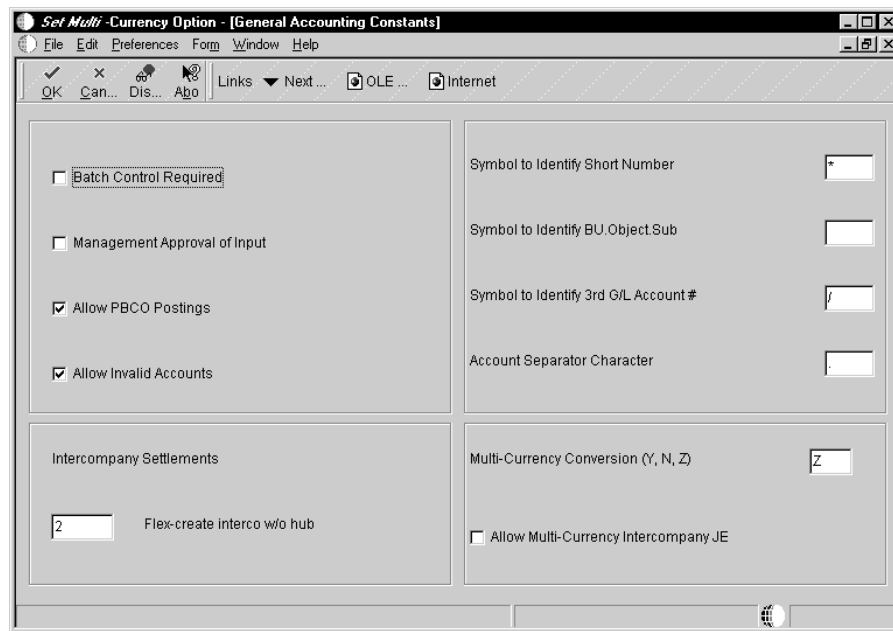
Setting Up Currency Conversion

▶ **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')

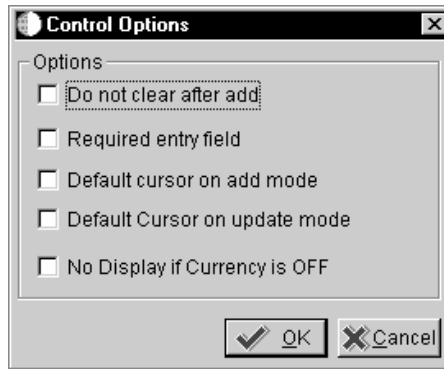
You can set the Multi-Currency Conversion option to “N” so that no currency conversion is done in JDB and the runtime engine.

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

1. On Forms Design, double-click the control that 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.

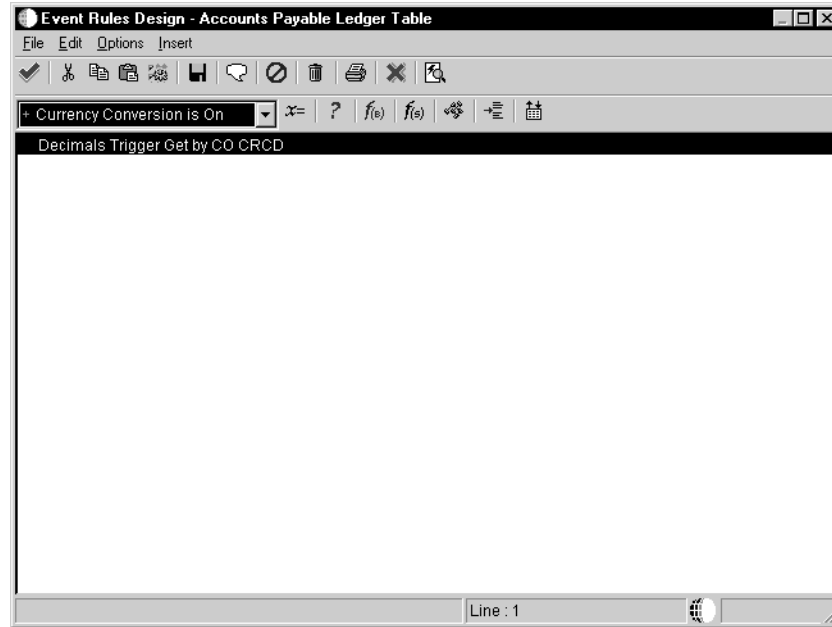
When currency is off, currency-sensitive controls are not displayed. If this option is turned off, currency fields are visible.

You must exit your current OneWorld session and re-enter in order to apply Currency Conversion changes.

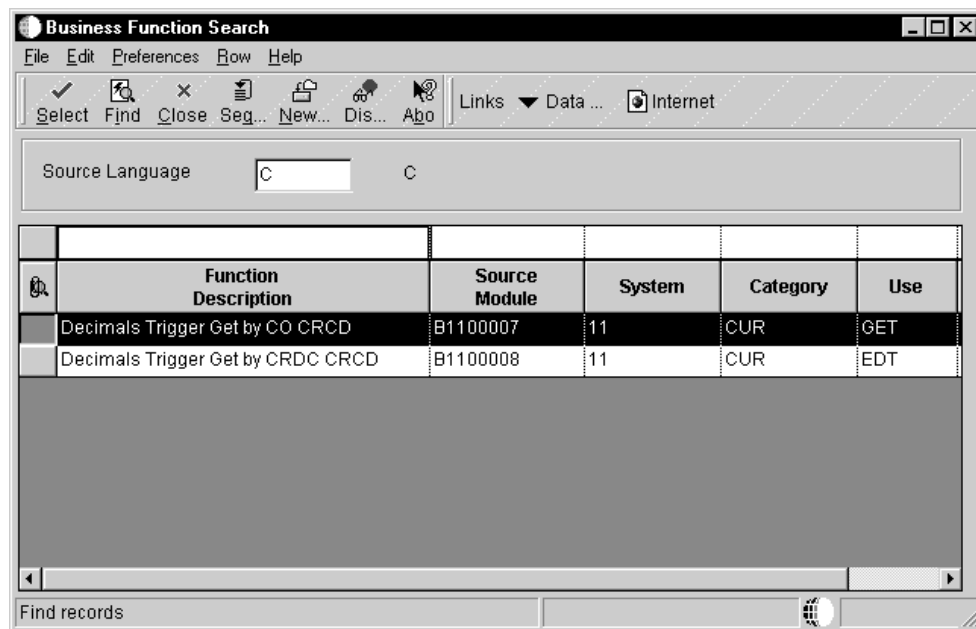
Creating a Currency Conversion Trigger

► To create a currency conversion trigger

1. From Object Management Workbench, check out the table to which you want to attach event rules.
2. Ensure the table is highlighted, and then click the Design button in the center column.
3. On Table Design, click the Design Tools tab and then click *Start Table Trigger Design Aid*.

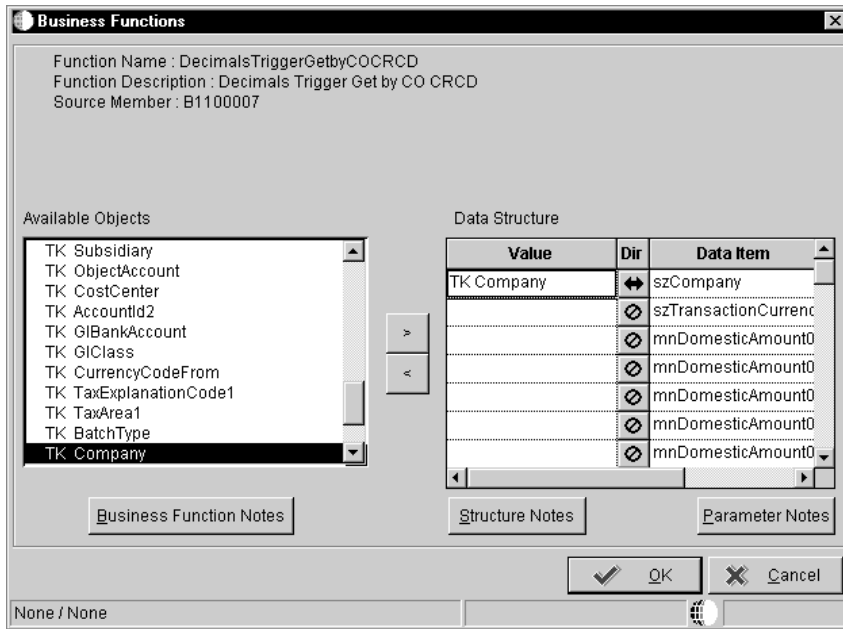


4. On Event Rules Design, choose the *Currency Conversion* event and attach the currency trigger that you want to use.
5. Click the Business Functions button. The Business Function Search form appears.



Use the QBE line to 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.

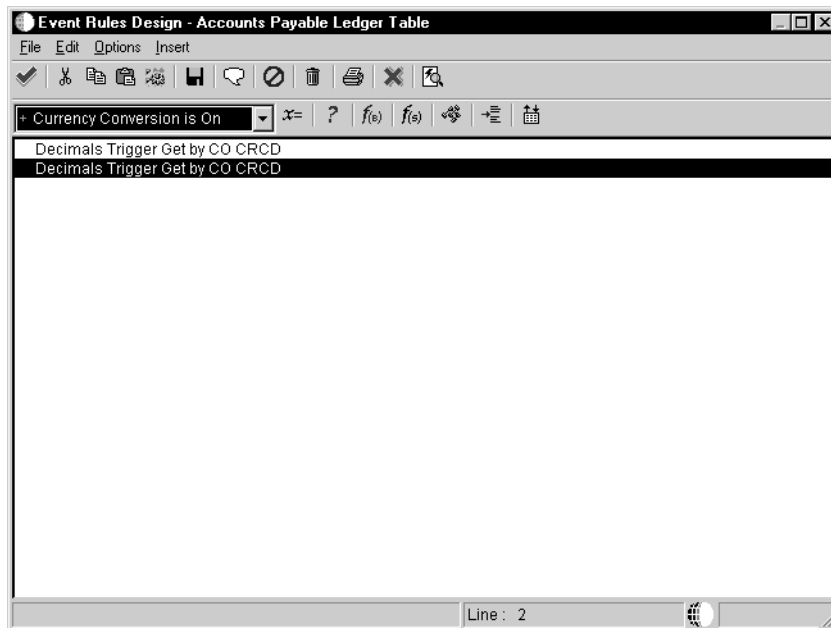
- Choose the business function with which you want to work, and then click Select.



- On Business Functions, attach the table columns to the business function structure, and then click OK.

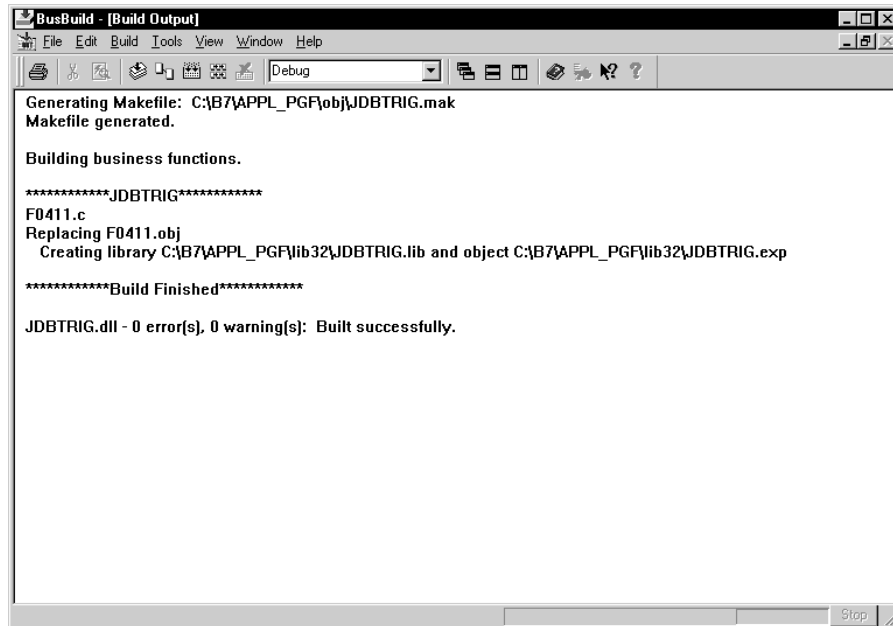
The available objects that appear are for table column only.

- On Event Rules Design, click Save, and then click OK.



9. On Table Design, click the Table Operations tab, and then click Generate Table.
10. Choose the Data Source for the table, and then click OK.
11. On Table Design, click the Design Tools tab, and then click Build Table Triggers.

The 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.



```
BusBuild - [Build Output]
File Edit Build Tools View Window Help
Debug
Generating Makefile: C:\B7\APPL_PGF\obj\JDBTRIG.mak
Makefile generated.

Building business functions.

*****JDBTRIG*****
F0411.c
Replacing F0411.obj
Creating library C:\B7\APPL_PGF\lib32\JDBTRIG.lib and object C:\B7\APPL_PGF\lib32\JDBTRIG.exp

*****Build Finished*****

JDBTRIG.dll - 0 error(s), 0 warning(s): Built successfully.
```




Menu Design

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

Menu design contains the following topics:

- Understanding menus
- 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 table (F0082) 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)

Understanding menus contains the following topics:

- Menu filtering
- Menu design tables

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 that the following selections 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.

Working with menus contains the following tasks:

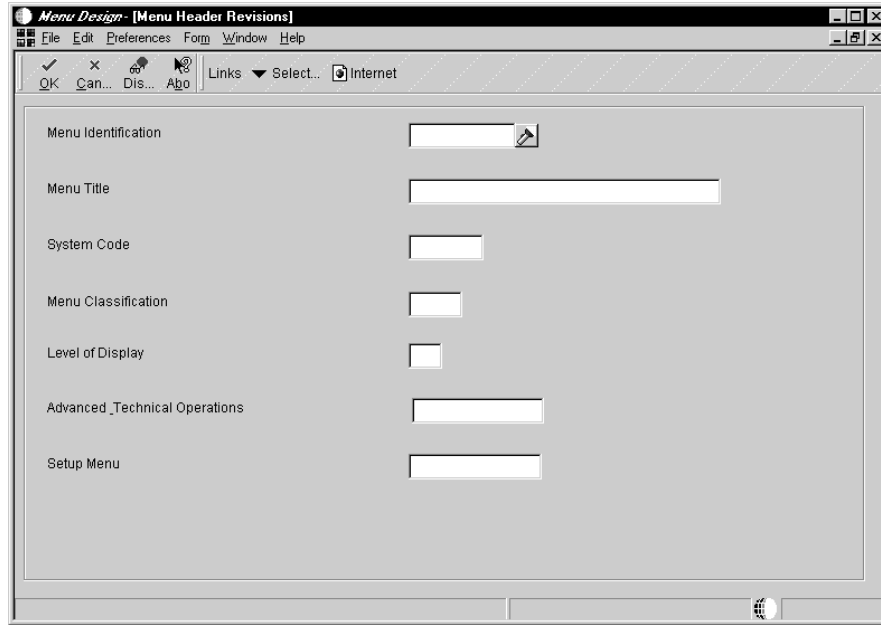
- Defining a new menu
- Reviewing selections for a menu
- Printing a menu report

Defining a New Menu

You can define a menu to include selections that enable you to access the applications and reports that you need from one location.

To define a new menu

1. On System Administration Tools (GH9011), 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	<p>The menu name, which can include up to nine characters. J.D. Edwards standards are:</p> <ul style="list-style-type: none"> • 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. <p>For example, the menu identification G0911 specifies the following:</p> <table style="margin-left: 20px;"> <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 Classification	The menu classification indicates the type of a menu. For example: a JDE Master menu or Company Master menu.
Level of Display	<p>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:</p> <ul style="list-style-type: none"> 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	<p>For World:</p> <p>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.</p> <p>..... <i>Form-specific information</i></p> <p>For OneWorld:</p> <p>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.</p>
Setup Menu	<p>For World:</p> <p>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.</p> <p>..... <i>Form-specific information</i></p> <p>For OneWorld:</p> <p>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.</p>

Reviewing Selections for a Menu

You can review the selections included on a specific menu from the Work With Menus form or from the Menu Header Revisions form.

► **To review selections for a menu**

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, locate and choose a menu that you wish to review.

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

Menu		Description		SY	LOD	Class				
DEMO		APPLICATIONS		00						
Sel #	Description	Job To	Form	Version	Ctry	Appl	Run Time	SC		
		Execute	Name	Ovrd	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.

Note: 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.

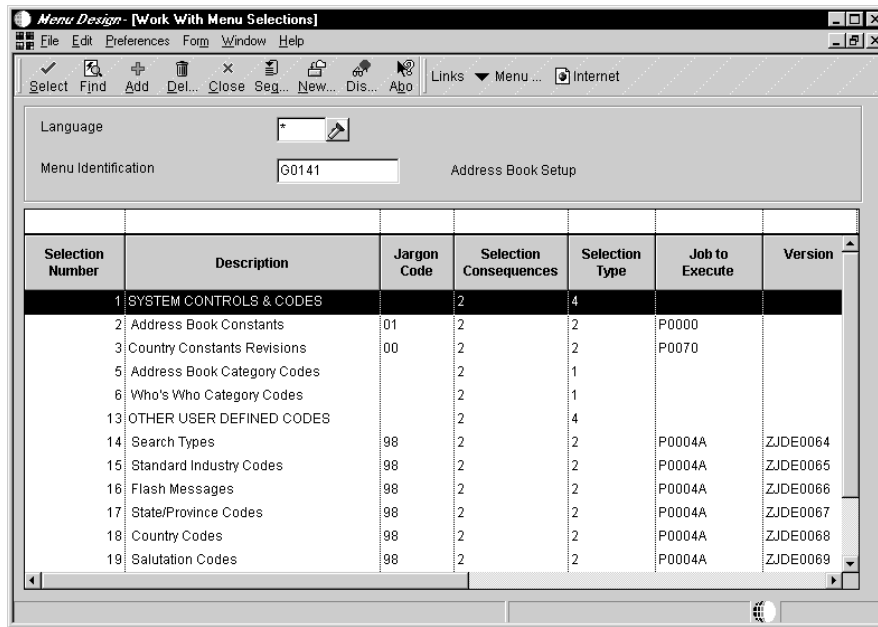
Adding or changing a menu selection consists of the following:

- Naming a menu selection

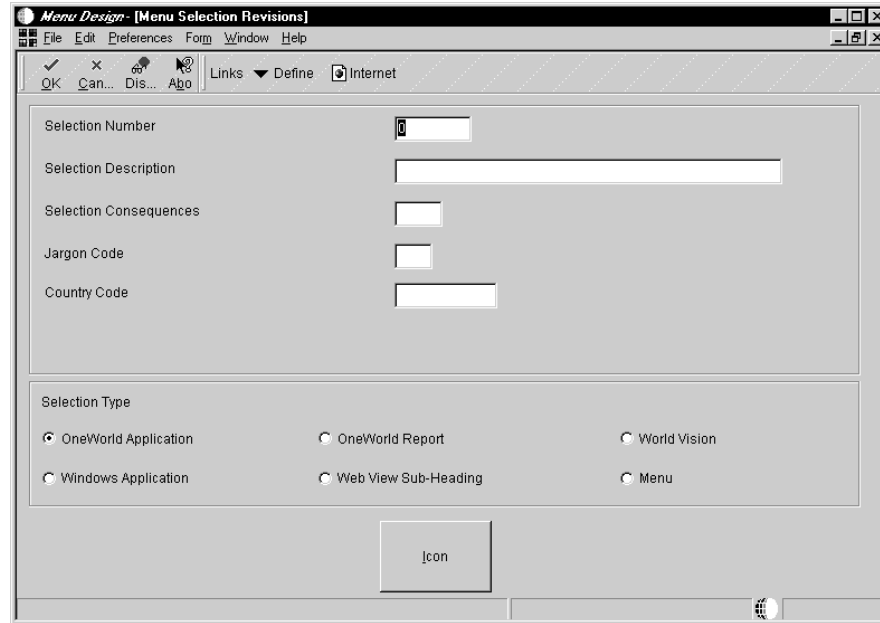
- Defining the menu selection

► **To name a menu selection**

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection that you want to name.



3. On Work With Menu Selections, click Add or choose an existing selection to change.



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 them 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

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want 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.

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

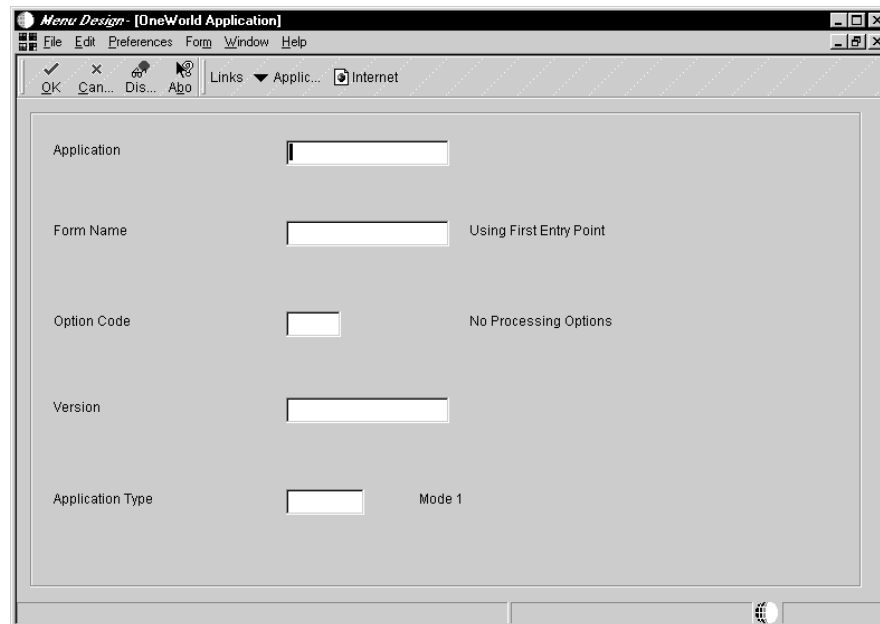
► **To add a OneWorld application to a menu**

You can use this procedure to add a OneWorld application created in Forms Design as a menu selection.

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want 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.



The screenshot shows a dialog box titled "Menu Design - [OneWorld Application]". The dialog has a menu bar with "File", "Edit", "Preferences", "Form", "Window", and "Help". Below the menu bar is a toolbar with buttons for "OK", "Cancel", "Dismiss", and "Apply", along with a "Links" dropdown menu and an "Internet" icon. The main area of the dialog contains several input fields and labels:

Application	<input type="text"/>	
Form Name	<input type="text"/>	Using First Entry Point
Option Code	<input type="text"/>	No Processing Options
Version	<input type="text"/>	
Application Type	<input type="text"/>	Mode 1

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	<p>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:</p> <p>Code</p> <table style="margin-left: 20px;"> <tr> <td style="padding-right: 10px;">1</td> <td>version — mandatory; option key — form i.d. F18 displays processing options. Selection = blind DREAM Writer execution.</td> </tr> <tr> <td>2</td> <td>version — blank; option key — form i.d. F18 displays DREAM Writer versions list. Selection = DREAM Writer versions list.</td> </tr> <tr> <td>2</td> <td>version — not blank; option key — form i.d. F18 displays DREAM Writer versions list. Selection = blind execution, batch.</td> </tr> </table> <p>Review the HELP instructions for Menu Information (Menu Locks) (P0090) for a detailed explanation of codes related to job submission and control.</p> <p>For OneWorld, this code specifies whether the user will be prompted for additional information prior to running the application. Available values are:</p> <table style="margin-left: 20px;"> <tr> <td style="padding-right: 10px;">0</td> <td>No processing options</td> </tr> <tr> <td>1</td> <td>Blind execution (no prompt)</td> </tr> <tr> <td>2</td> <td>Prompt for version</td> </tr> <tr> <td>3</td> <td>Prompt for values</td> </tr> </table>	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.	0	No processing options	1	Blind execution (no prompt)	2	Prompt for version	3	Prompt for values
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.														
0	No processing options														
1	Blind execution (no prompt)														
2	Prompt for version														
3	Prompt for values														
Version	<p>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.</p>														

Field	Explanation
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.

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want 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.
6. On OneWorld Report, complete the following fields:
 - Batch Application
 - Version
7. Select the processing options you want users to have:
 - Blind Execution
 - Values
 - Versions
 - Data Selection

Use Form menu options to view and choose from a list of reports and versions.

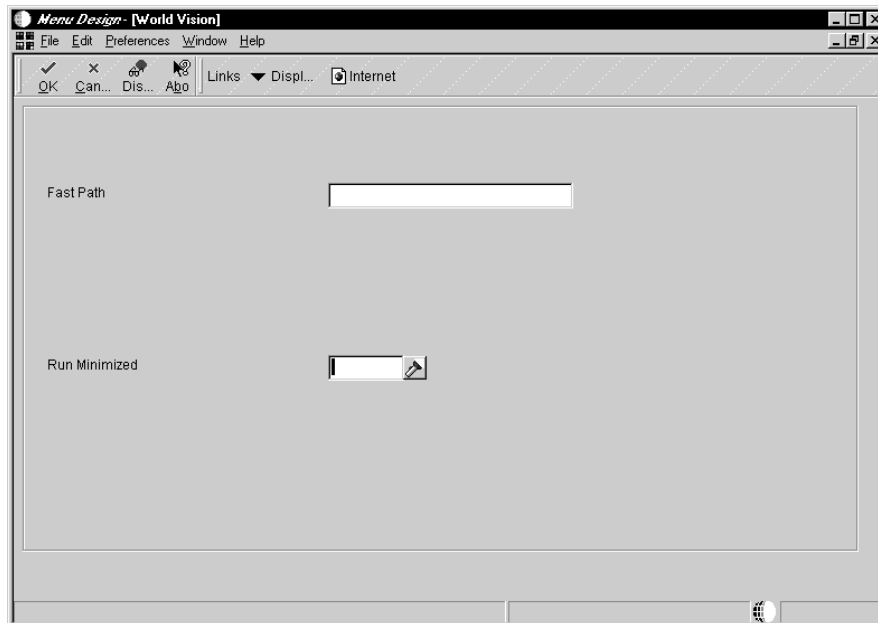
► To add a WorldVision application to a menu

You can add a WorldVision application as a menu selection.

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want 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	<p>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.</p> <p>To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.</p>
Run Minimized	<p>The Run Minimized flag determines if a Windows application is to be minimized to an icon when you open it.</p>

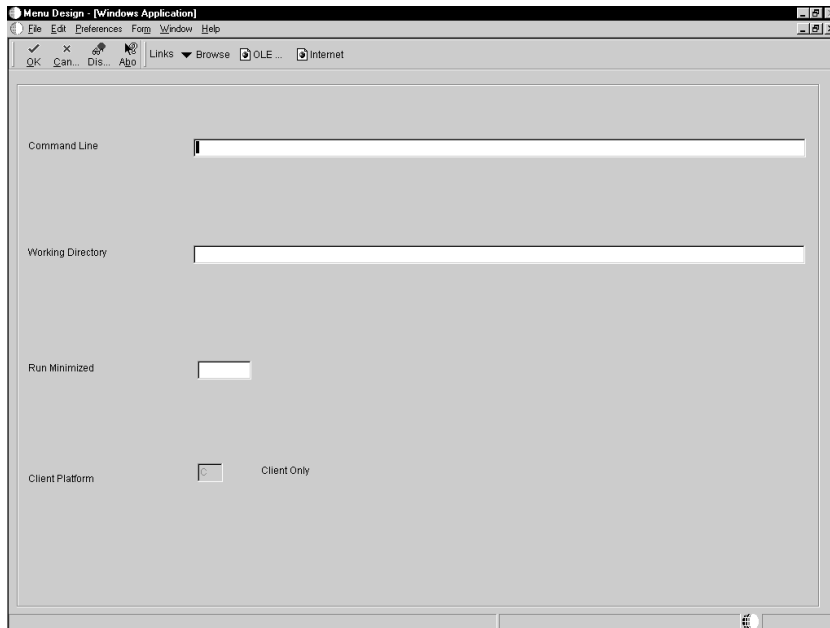
To add a Windows application to a menu

You can add any Windows applications as menu selections. For example, you can add Windows programs such as Calendar, Clock, Note Pad, or Write.

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want 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 that 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	<p>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.</p> <p>To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.</p>

Field	Explanation
Working Directory	<p>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.</p> <p>To specify the path for a World Vision menu selection, the path includes the selection number, slash, menu.</p>

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

1. On System Administration Tools (GH9011), 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

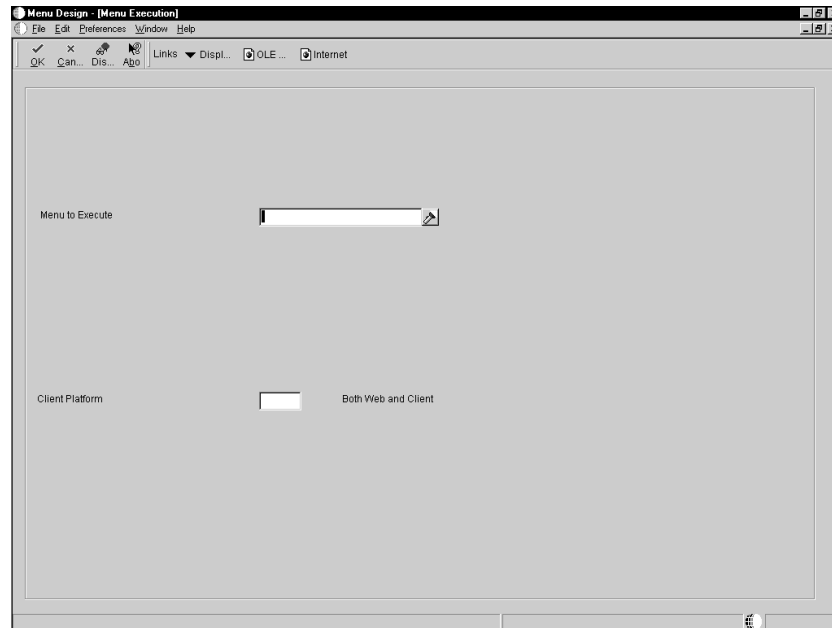
1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. Find and choose the menu for which you want to create a subheading.
3. On Work With Menu Selections, click Add.
4. On Menu Selection Revisions, complete the following field:
 - Selection Number
5. Click the Web View Sub-Heading 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

1. On System Administration Tools (GH9011), Choose Menu Design (P0082).
2. Find and choose the menu for which you want 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
 - Client Platform

Field	Explanation
Menu to Execute	The specific menu to be executed as a selection on a menu.

Creating Fast Path Selections

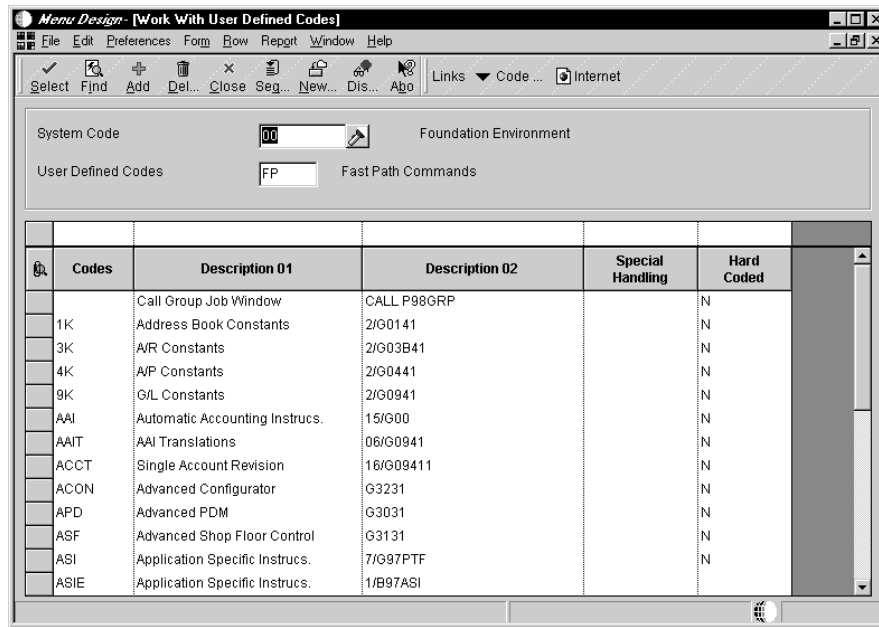
You can quickly move among menus and applications by using fast path commands. A fast path command is:

- 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 that 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.

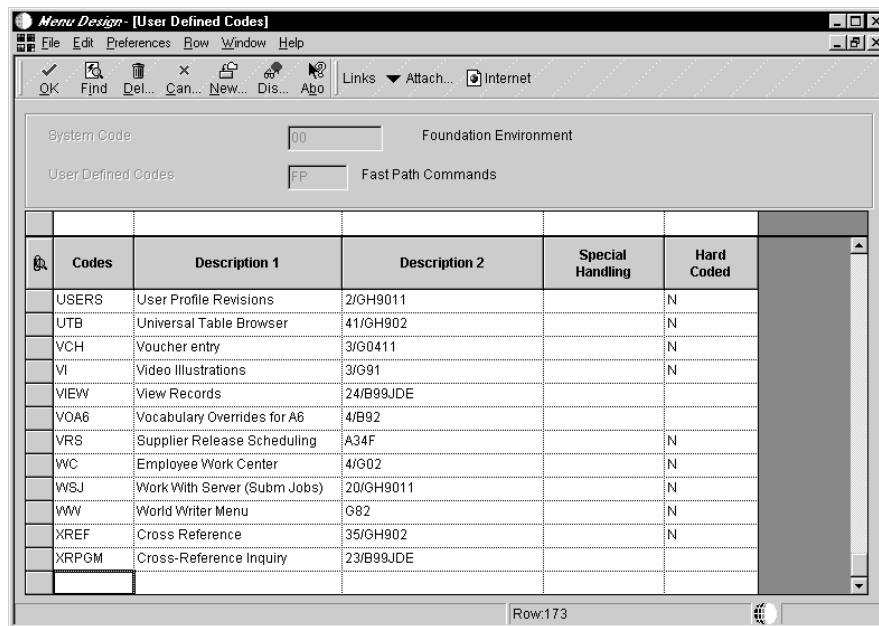
You can set up your own a fast path abbreviations to access frequently used applications.

► **To create a fast path selection**

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection for which you want to create a fast path.
3. On Work With Menu Selections, from the Form menu, choose Fast Path Revs.



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:

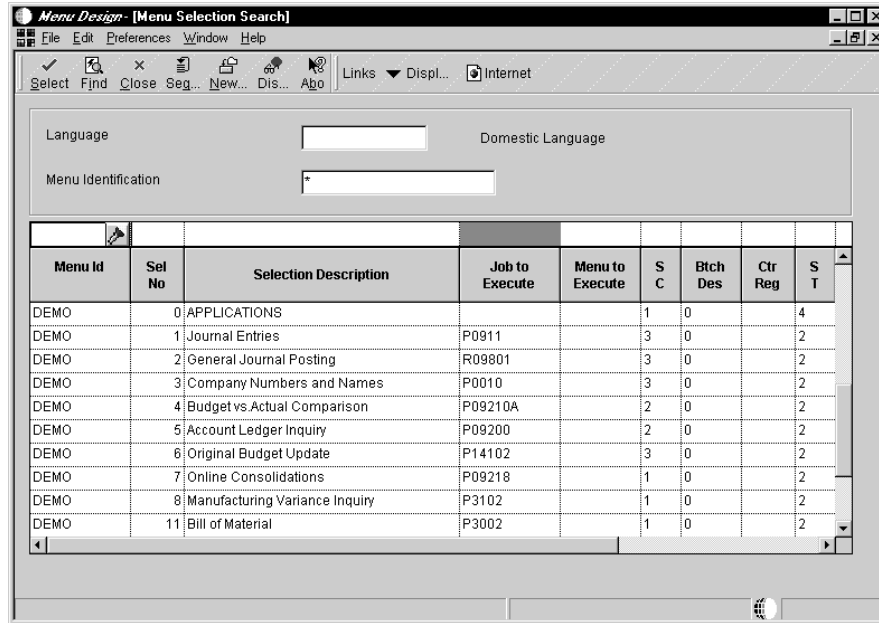
- Copying a menu selection
- Changing menu text for languages
- Changing menu selection text
- Renumbering a menu selection

Copying a Menu Selection

You can copy an existing menu selection and attach it to another menu.

To copy a menu selection

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu that has a selection you want 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.



5. 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

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu for which you want to change menu text.
3. On Work With Menus, choose Header from the Row menu selection.

4. On Menu Header Revisions, from the Form menu, choose Title Overrides.

Language	Menu Text

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

1. On System Administration Tools (GH9011), choose Menu Design (P0082).
2. On Work With Menus, find and choose the menu for which you want 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.

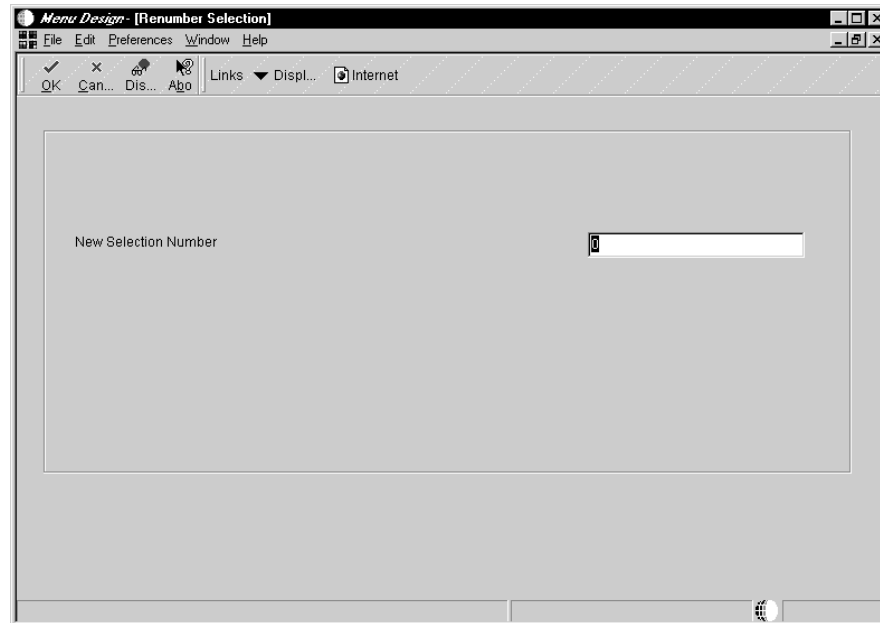
Renumbering a Menu Selection

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

1. On System Administration Tools (GH9011), choose Menu Design (P0082).

2. On Work With Menus, find and choose the menu for which you want to change the menu's selection number.
3. On Work With Menu Selections, choose the menu selection that 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.



Tips of the Day

Tips of the day are a sets of short, informational text that appear in a special form each time the user launches an application or accesses a form. Tips of the day appear sequentially, so the user can browse through the tips. When the user closes the tip form, the system records where in the tip sequence the user is and displays the next tip when the user launches the object again.

J.D. Edwards provides tips of the day with many OneWorld applications. You can change these tips sets or create your own. Tips of the day contains the following topic:

- Working with tips of the day



Working with Tips of the Day

In OneWorld, tips of the day are the glossary texts of data dictionary items. You create one data dictionary item for each tip. Since you can translate data dictionary glossaries, tips of the day can appear in different languages.

You can associate tips with an application, a form, or an application version. The tips appear in the order you specify, and you can override a user's option to turn off the tip of the day feature for the tip set.

After you have associated tips with an object, you can rearrange the tip order, add new tips, or delete existing ones from the tip set.

Working with Tips of the Day contains the following tasks:

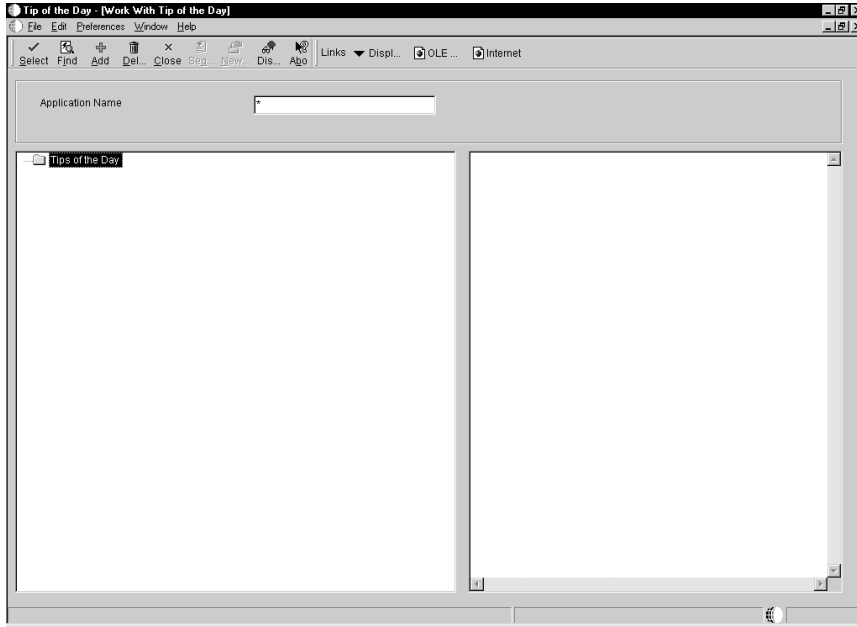
- Working with the Tip of the Day utility
- Adding tips of the day to an object

Before You Begin

- Create a data dictionary item with glossary text for each tip of the day. See *Creating a Data Item* for detailed instructions about creating data dictionary items.

▶ **To work with the Tip of the Day utility**

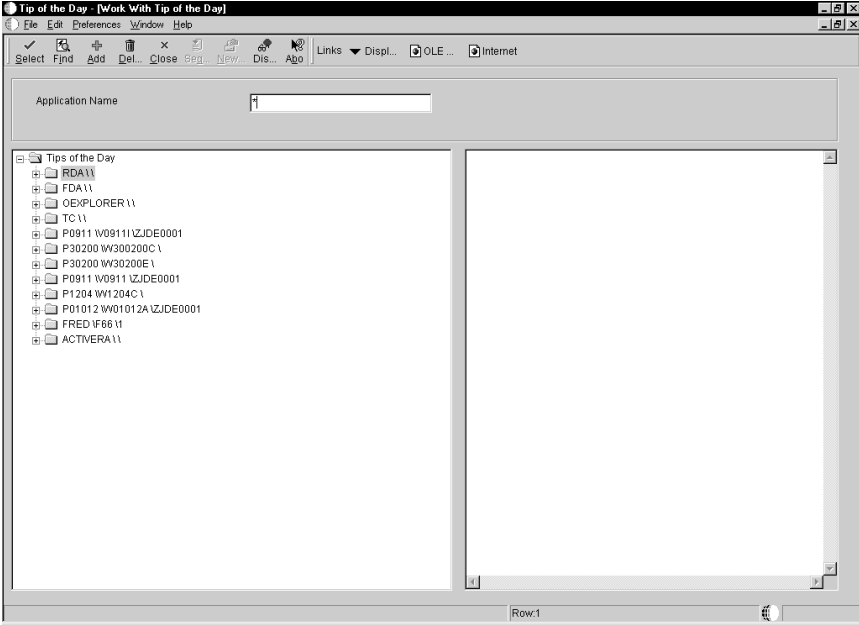
1. From the System Administration Tools menu (GH9011), double-click Tip of the Day.



2. On Work With Tip of the Day, click Find to see the objects to which tips have been added.

Enter an application name in the Application Name field to limit your search.

A tree-style file structure appears on the left side of the form. Each object has its own folder. Tips can be associated with applications, forms, or application versions. Folder names include the application, form, and application version, in that order. For example, RDA\\ indicates the RDA application alone. P0911\V0911I\ZJDE0001 indicates form V0911I in the ZJDE0001 version of application P0911.

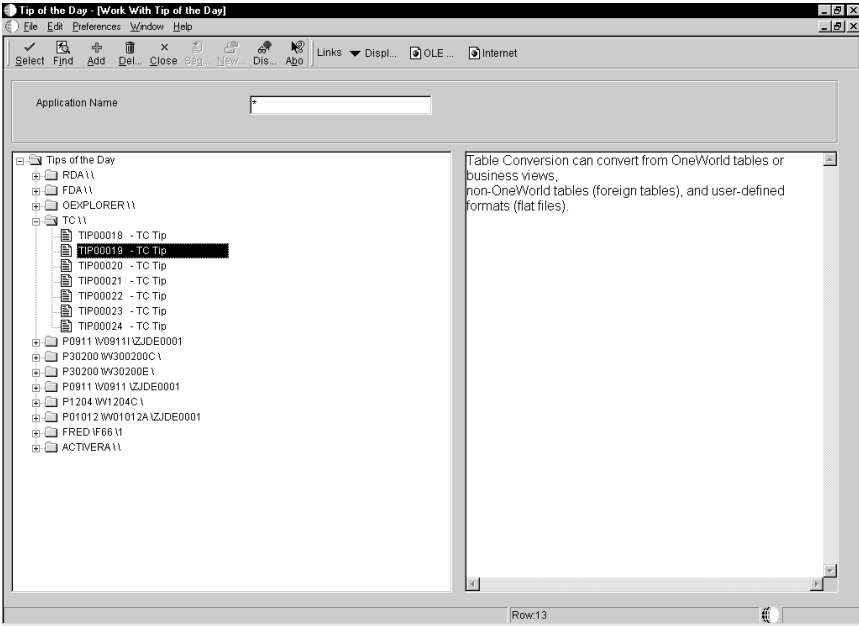


- 3. To see the specific tips associated with an object, expand the folder for the object.

The data dictionary items associated with the object as tips appear in the file structure.

- 4. To view the text for a tip, click the tip.

The glossary text associated with the data dictionary item appears on the right side of the form.



5. To change the order of the tips or to add or delete tips for an object that already has tips associated with it, double-click one of the tips under the object.

The Tips of the Day Revisions form appears.

To add tips of the day to an object

Note: Use this task to add tips to an application, form, or application version that does not already have tips associated with it. To add tips to an existing tip set, double-click one of the existing tips on the Work With Tip of the Day form to access the Tips of the Day Revisions form.

1. On Work With Tip of the Day, click Add.
2. On Tips of the Day Revisions, complete the following fields:
 - Application Name
 - Form Name
 - Version
 - Description
 - Force tip to all users
3. In the detail area, add a data dictionary item for each tip you want to associate with the application. Complete the following columns for each row:
 - Tip Sequence
 - Data Item

Note: You cannot add a data dictionary item to the tip set if it does not have glossary text.

4. When finished adding data dictionary items to the application, click OK.

Field	Explanation
Application Name	Tip of the day application.
Form Name	The unique name assigned to a form.

Field	Explanation
Version	A user-defined set of specifications that 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	Describes the tip of the day application.
Force tip to all users	Allows the tips for an application to be forced on.
Tip Sequence	The tip of the day sequence.
Data Item	<p>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.</p> <p>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 #, @, \$.</p> <p>You can create protected data names by using \$xxx and @xxx, where you define xxx.</p> <p>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: % & , . +.</p> <p>Create new data items using system codes 55-59.</p> <p>The alias cannot be changed.</p>

