# JD Edwards EnterpriseOne Tools

## Event Rules Guide

9.2

JD Edwards EnterpriseOne Tools
Event Rules Guide

9.2

Part Number: E53554-03

Copyright © 2011, 2022, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

# Preface

Welcome to the JD Edwards EnterpriseOne documentation.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc* .

## Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info* or visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs* if you are hearing impaired.

## Related Information

For additional information about JD Edwards EnterpriseOne applications, features, content, and training, visit the JD Edwards EnterpriseOne pages on the JD Edwards Resource Library located at:

*http://learnjde.com*

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
|---|---|
| **Bold** | Boldface type indicates graphical user interface elements associated with an action or terms defined in text or the glossary. |
| *Italics* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `Monospace` | Monospace type indicates commands within a paragraph, URLs, code examples, text that appears on a screen, or text that you enter. |
| **> Oracle by Example** | Indicates a link to an Oracle by Example (OBE). OBEs provide hands-on, step- by-step instructions, including screen captures that guide you through a process using your own environment. Access to OBEs requires a valid Oracle account. |

# 1 Introduction to JD Edwards EnterpriseOne Tools: Event Rules

## JD Edwards EnterpriseOne Tools: Event Rules Overview

Oracle's JD Edwards EnterpriseOne Tools for event rules are used to create or modify event rules (ER) in JD Edwards EnterpriseOne applications. Event rules are connected to certain runtime events and instruct runtime how to respond to the conditions you choose to define.

## JD Edwards EnterpriseOne Tools: Event Rules Implementation

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Tools for event rules.

In the planning phase of your implementation, take advantage of all JD Edwards EnterpriseOne sources of information, including the installation guides and troubleshooting information.

### JD Edwards EnterpriseOne Tools Development Tools: Event Rules Implementation Steps

The following steps need to be performed before working with JD Edwards EnterpriseOne event rules:

1. Configure Oracle's JD Edwards EnterpriseOne Tools Object Management Workbench (OMW).

   See *"Configuring JD Edwards EnterpriseOne OMW" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .
2. Configure OMW user roles and allowed actions.

   See *"Configuring User Roles and Allowed Actions" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .
3. Configure OMW functions.

   See *"Configuring JD Edwards EnterpriseOne OMW Functions" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .
4. Configure OMW activity rules.

   See *"Configuring Activity Rules" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .
5. Configure OMW save locations.

   See *"Configuring Object Save Locations" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

6. Set up default location and printers.

   See *"Understanding Report Printing Administration Technologies" in the JD Edwards EnterpriseOne Tools Report Printing Administration Technologies Guide* .

# 2 Understanding Events, Event Rules, and Runtime Processing

## Events

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

## Event Rules

This section discusses:

- Event rules
- Named event rules
- Embedded event rules

### Event Rules Fundamentals

*Event rules* (ER) are logic statements that you can create and attach to events. ER is initiated when events occur at runtime. Oracle's JD Edwards EnterpriseOne software supports two kinds of event rules: named event rules and embedded event rules.

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

- Conditional statements, such as If/Else/End If.
- While loops.
- Assignments.
- Calls to business functions.
- Form or report interconnections.
- Calls to system functions.
- Table I/O operations.

### Named Event Rules

A *named event rule* (NER) is a series of regular ER statements (such as assignments, business functions, system function calls, and so forth). A NER encapsulates the series of statements into one reusable component. You can call a NER the same way as calling a business function. Business functions implement customized business logic using C language; NERs implement customized business logic using event rule statements.

# Embedded Event Rules

In addition to NERs, the other kind of ER is called *embedded event rules*, or simply *event rules*. Embedded ER is specific to a particular table, interactive application, or batch application. Embedded ER for a table is called *table event rules* or *table triggers*. Embedded ER for an interactive application or batch application is called *application event rules*.

## Application Event Rules

You can add business logic that is specific to a particular application. Interactive applications connect ER using Oracle's JD Edwards EnterpriseOne Form Design Aid (FDA), while batch event rules use Oracle's JD Edwards EnterpriseOne Report Design Aid (RDA).

## Table Event Rules

You can create table-specific event rules, which are ER that you attach to a table using Table Design Event Rules. This logic runs whenever any JD Edwards EnterpriseOne application accesses that table *and* uses that ER. For example, to maintain referential integrity, you might attach ER to a master table that deletes all children when a parent is deleted. Any JD Edwards EnterpriseOne application that deletes information from that table does not need to have embedded parent/child logic, because that logic exists in the table.

> **Note:** Be aware that this functionality applies *only* to JD Edwards EnterpriseOne applications. Other applications that access the same database table cannot and do not use these ERs.

# Runtime Processing of Event Rules

This section discusses:

- Runtime processing of event rules.
- Runtime data structures.
- Form flow.

# Fundamentals of Runtime Processing of Event Rules

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

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

To avoid generating unnecessary ER, you should understand the different field types and associated capabilities that characterize each form type.

# Runtime Data Structures

*Runtime data structures* are structures or blocks of memory that hold data when user is working with an application. You should know what is happening to each form at runtime. You should know what is in a runtime structure at a given event point in the runtime process.

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

## Available Objects and Runtime Data Structures

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

During runtime processing, the system stores data in memory in an internal data structure. Certain fields of the data structure temporarily store data during runtime. When no longer needed, the data is deleted so that the system can process another record.

In ER, you can access and modify available objects in order to implement business logic. For example, you can assign a value to a QBE field to set query criteria for the form.

This table lists the available objects:

| Available Object Code | Description |
|---|---|
| BC | A column in the business view (BV). BCs for both the form view and the grid view appear in this list. The system fills these columns with values from the database when it performs a fetch. The system writes these values to the database during an add or update. |
| GC | A column in the grid. The row that the value references depends on which event is accessing the GC. During the fetch cycle, it is usually the selected row. In some circumstances, CG objects also denote a particular physical column in the grid instead of a value. An example is the Set Grid Font system function. |
| GB | The grid buffer. This buffer is one row of data that is independent of the lines that the system reads from the database and writes to the grid. The GB enables you to manipulate column data for a line that you want to insert or update without affecting the present state of the grid. You access the GB through an available GB object, which appears after the GC objects in the list of available objects in Event Rules Design. Each grid contains only one instance of each GB column. |
| FC | A control on the form. If the control is a database item, this field corresponds to a BC object. Furthermore, if the control is not a filter, the FC object represents the same value as the BC object, and changing one of these results in changing both. |
| FI | A value passed through a form interconnection. You access this object either to read values that are passed into the form or to set values to be passed back. These objects correspond to the elements of the form data structure. |

| Available Object Code | Description |
|---|---|
| PO | A value passed from a PO. These values are passed into the application when a user launches it. Any form in that application can access them. POs can either be entered by the user, or they can be set up in a particular version of an application. |
| QC | A cell from the QBE line in the grid. These objects represent the values in any QBE cell on the grid. They include wild cards, but do not include any comparison operators. Likewise, assignments to these objects can include wild cards, but not comparison operators. To set comparisons, you must use a system function. |
| HC | A hypercontrol item. A hypercontrol item is a menu item or a tool bar item. |
| VA | ER variables. These objects represent any variables that you set up in ER. |
| SV | System variables. These objects represent some environment variables that are accessible to ER. |
| SL | System literals. These objects represent some constant system values that are accessible to ER. |
| TP | Tab page object. |
| TK | A column in the table that contains the table ER. |
| CO | A constant, such as the return code for an error. |
| TV | Text variables. |
| RC | Report constants for a batch application. |
| RV | Report variables (batch application). |
| IC | An input column (table conversion). |
| OC | An 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.

## Processing Available Objects

When an available object is changed through ER, these actions occur:

- The object in the internal runtime structure is changed.
- If the object is a form control or grid cell, row, or column, the screen is updated with the new value.

A BV form control shares the same value as the corresponding business view item. (Filter fields are an exception to this rule.) This means that:

- FC data and BC data are always identical.

- Whenever FC data is changed, BC items are changed to the same value.
- Whenever BC values are changed, the FC runtime values also changes to the same values. This change may not immediately reflected to the screen.
- On Control is Exited processing, the value entered into the form control is captured in both the BC and FC item for that control.

### Control is Exited Processing

Control is Exited processing includes these actions:

- The value in the control is saved to internal runtime structures.
- The Control is Exited event is processed.

If the value has changed since the previous time that the control was exited, these steps occur:

- The system processes the Control Exited/Changed–Inline event.
- The system processes the Control Exited/Changed–Async event.
- The system validates the value using edit rules defined for the DD item.
- The form control data is formatted using format rules defined for the DD item and displayed on the screen.

The Trigger Parallel Event system function is available for the **Control Exited/Changed–Inline** and the **Control Exited/Changed–Async** events. This system function will enable a parallel event to run on a separate thread and will not interfere with existing Event Rules.

## Form Flow

Each form type has different properties and event flow. The system provides events for the forms so that you can insert custom logic. These events occur regardless of whether you add event rule logic for that event.

This 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 next sections describe the typical events for the find/browse form and the order in which they are processed. This process flow can vary depending on specific user interaction and the event rule logic that you use.

### Pre-Dialog Is Initialized

These steps occur before the Dialog is Initialized event is processed and the form appears:

- Initialize runtime structures (or clear memory) as shown:
  - BC = null.
  - FC = null.
  - GC = null.
  - FI = Values passed from a calling form (if any).
  - PO = Values passed from processing options.
- Initialize form controls.
- Initialize error handling.

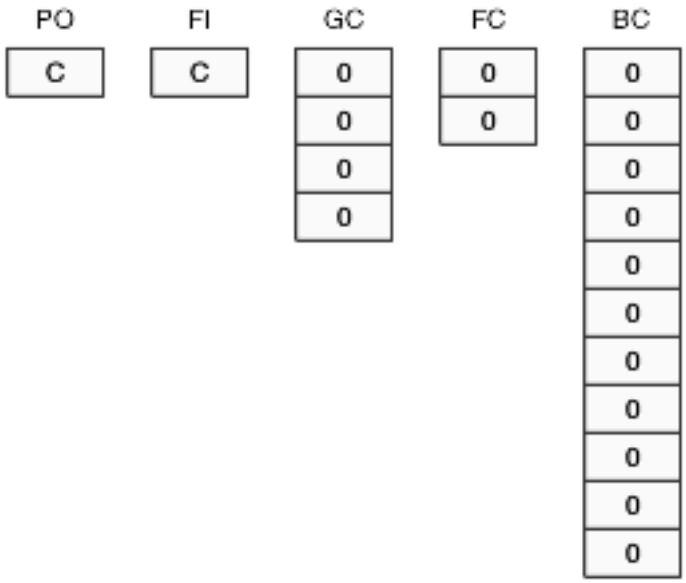
- Initialize static text.
- Initialize helps.
- Create tool bar.
- Load form interconnect data into corresponding BC columns and filter fields (if any exist).
- Initialize thread handling.

## Dialog Is Initialized

The system processes all event rule logic that is attached to the Dialog is Initialized event. When this event starts, the runtime structures contain these values:

- BC = Any FI values passed.
- FC = Any FI values passed.
- GC = null.
- FI = Values passed from a calling form (if any).
- PO = Values passed from POs.

This diagram illustrates the information in the runtime structures just before the system fires Dialog is Initialized:



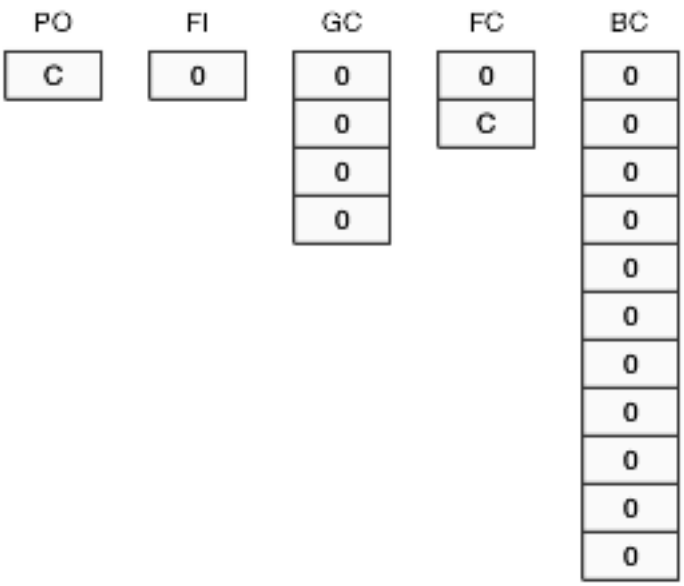

The Dialog is Initialized event can be used to initialize the form. After the Dialog is Initialized event is finished, runtime starts the Post Dialog Is Initialized process.

## Post Dialog Is Initialized

Before the system fires the Post Dialog is Initialized event, the runtime structures contain these values:

- BC = null (or values already passed in).
- FC = null (or values already passed in).
- GC = null (or values already passed in).
- FI = Values passed from a calling form (if any).
- PO = Values passed from POs.

This diagram illustrates the information in the runtime structures just before the system fires the Post Dialog Is Initialized event:



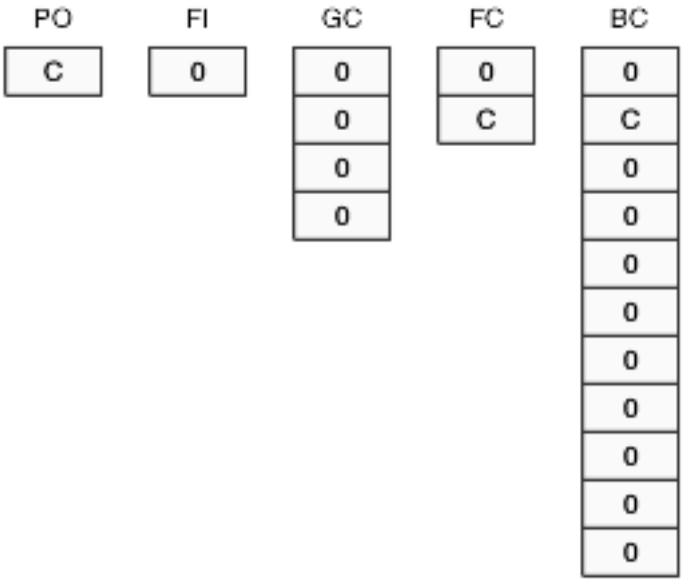

The Post Dialog is Initialized event is commonly used to perform these tasks:

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

## Building SQL SELECT

After the user clicks Find, the system builds a `SELECT` statement with a `WHERE` clause. The `SQL SELECT` statement includes all columns in the BV. The `WHERE` clause includes any values in the QBE or filter fields. It can also contain values passed through Set Selection and Set Lower Limit system functions. The `WHERE` clause is then used to get all records that meet the criteria.

This diagram illustrates the information that appears in the runtime structures just before the system builds the SQL statement:




## Fetching Records

Records are fetched one page at a time (unless page-at-a-time is disabled). The system processes each record fetched one by one and display it in the grid row.

## Page-at-a-Time Processing

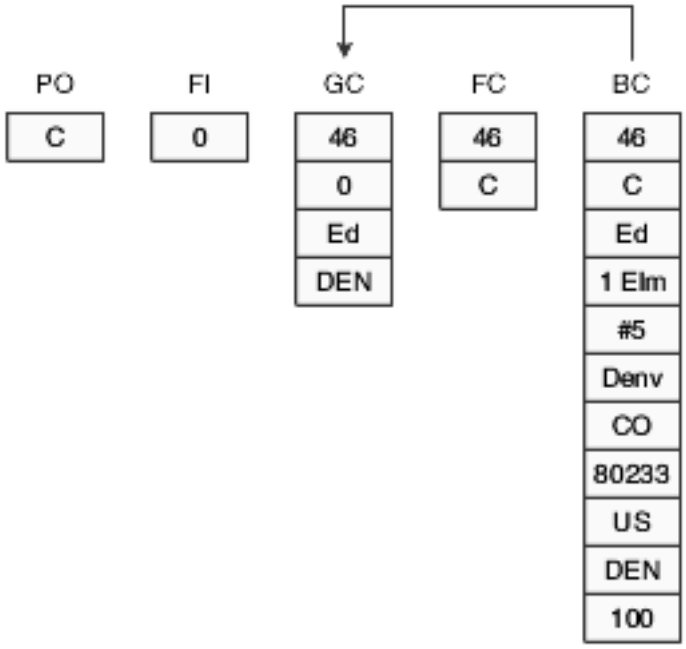
Page-at-a-time processing means that the system fetches only a single page worth of records to display. To see the next page of records, the user clicks the Next button. You can customize the page size for each grid in FDA. A system administrator can also set a global page size for all grids.

Typically, page-at-a-time processing improves performance and scalability. Although it can be disabled, the JD Edwards EnterpriseOne standards state that you should not disable it unless you have a valid business reason to do so.

## BC Assigned Database Values

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

This diagram illustrates the information in the runtime structures when the system reads the first record:

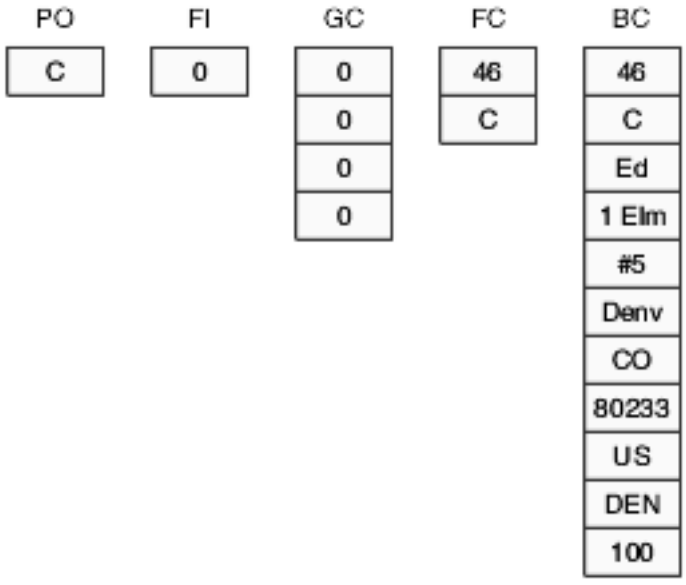



## Grid Record Is Fetched

The engine then fires the Grid Record is Fetched event. At this point, the runtime structures have these values:

- BC = Values from the database (for the first record read).
- FC = Values from the database (if the fields are database fields).
- GC = null.
- FI = Values passed from a calling form (if any).
- PO = Values passed from POs.

This diagram illustrates the information in the runtime structures just before the system fires Grid Record is Fetched:

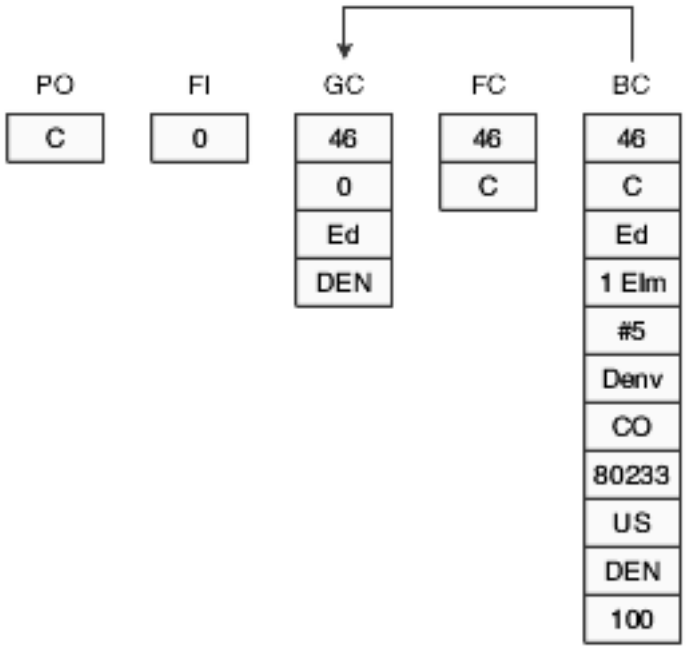


The Grid Record is Fetched event is commonly used to perform these actions:

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

After the Grid Rec Is Fetched event fires, the BC values are copied into the GC runtime structure.

This diagram illustrates the information in the runtime structures when the system reads the first record:

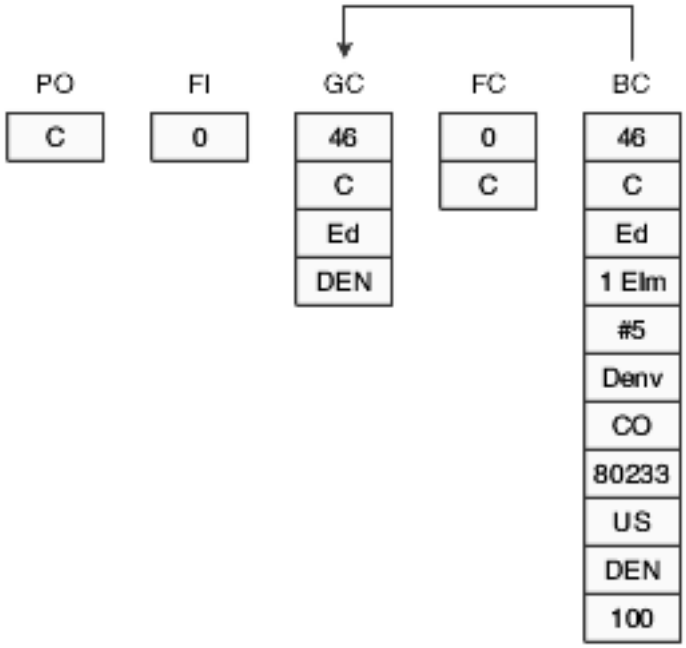



## Write Grid Line-Before

The engine then fires the Write Grid Line-Before event. A this point, the runtime structures have these values:

- BC = Values from the database (from the record just read).
- FC = Values from the database (if the fields are database fields).
- GC = Values from the database (from the previous read).
- FI = Values passed from a calling form (if any).
- PO = Values passed from POs.

This diagram illustrates the information in the runtime structures just before the system fires Write Grid Line-Before:

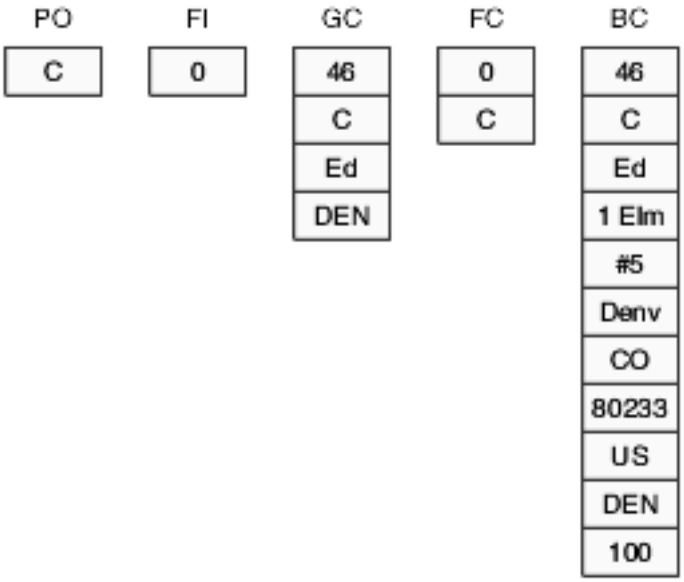



The Write Grid Line–Before event is commonly used to perform these tasks:

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

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

This diagram illustrates the information that appears in the runtime structures now:



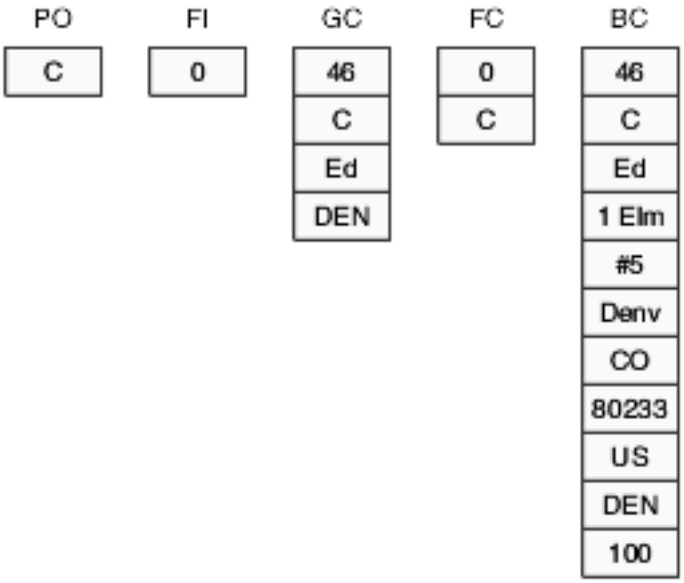

## Write Grid Line–After

The engine then fires the Write Grid Line-After event. At this point, the runtime structures have these values:

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

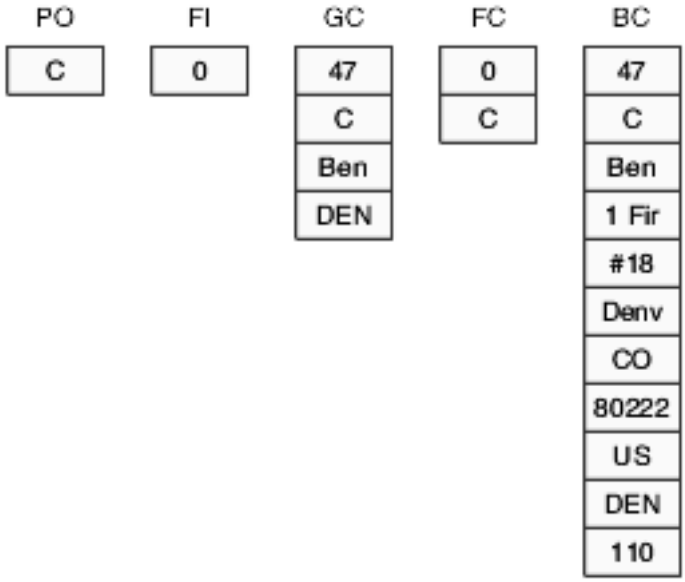
The system displays the current record in the grid cells.

This diagram illustrates the information in the runtime structures just before the system fires Write Grid Line–After:



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

This diagram illustrates the information in the runtime structures after the system processes Write Grid Line–After:




The system continues to read records from the database and performs the same processing steps. When the system reads the next record, it performs these processing steps:

- Assign BC values from the database.
- Process Grid Rec is Fetched ER.
- Assign BC values to GC.
- Process Write Grid Line–Before ER.
- Display values in the grid row on the form.
- Process Write Grid Line–After ER.

This process is repeated until there are no more records fetched.

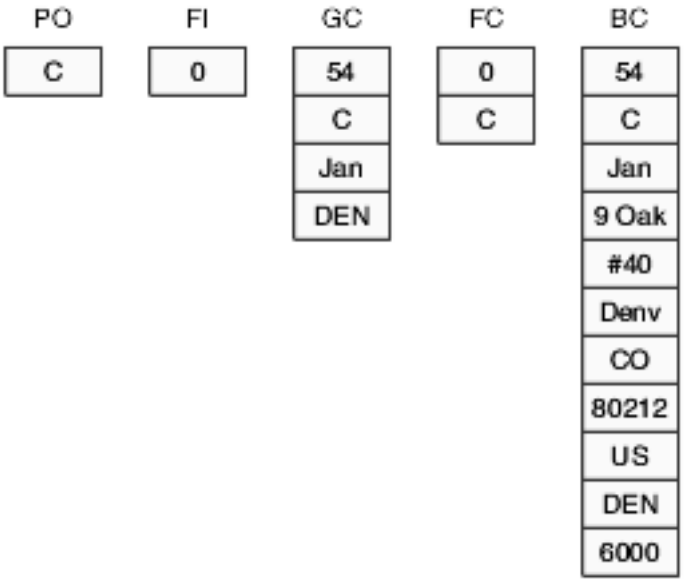
## Last Grid Record Has Been Read

When there are no more records fetched from the database, the engine fires Last Grid Record Has Been Read event. At this point, the runtime structures contain these values:

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

The GC values appear on the last grid row.

This diagram illustrates the information in the runtime structures just before the system runs Last Grid Record Has Been Read:

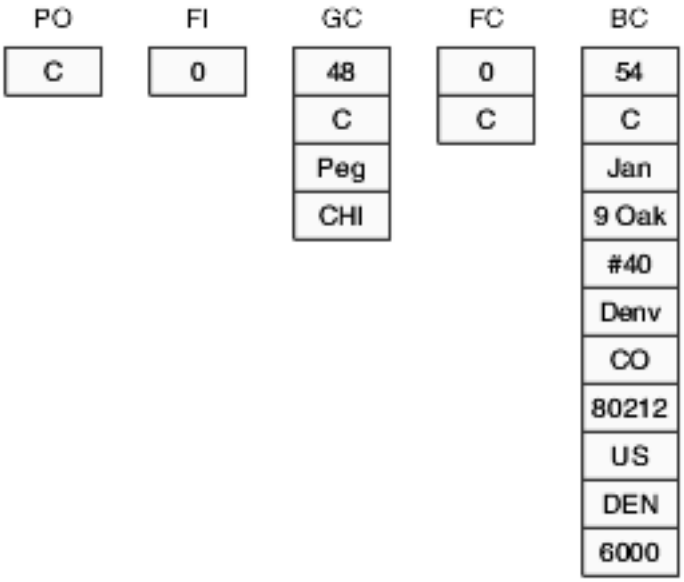



The Last Grid Record Has Been Read event is commonly used to write total lines to the grid and to display totals that are based on grid values.

## Select Button Processing

When a user selects a grid row and clicks the Select button, the BC structure stays the same, however the GC structure reflects values on the row that is being selected.

This diagram illustrates the information in the runtime structures when the user selects a grid row that is other than the last fetched row. Note that the BC and GC structures do not contain the same values:

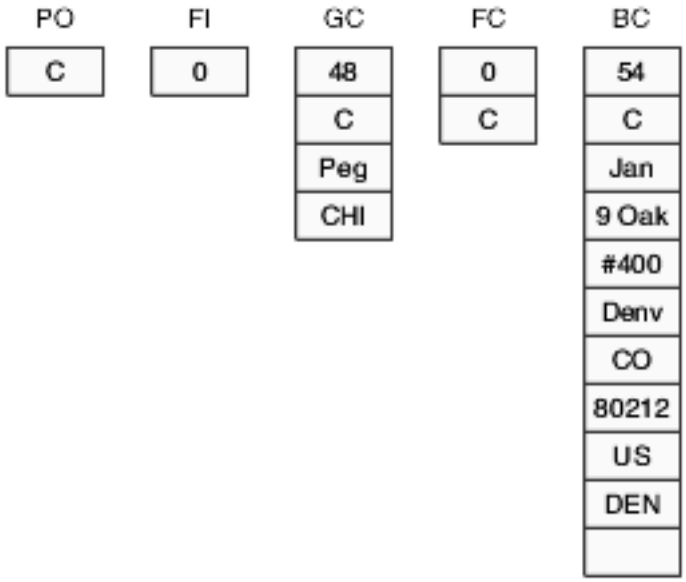


## Button Clicked

The engine then fires the Button Clicked event for the Select button. At this point, the runtime structures have these values:

- BC = Values from the database (from the last record read).
- FC = Values from the database (if the field is a database field).
- GC = Values from the selected grid row.
- FI = Values passed from a calling form (if any).
- PO = Values passed from processing options.

This diagram illustrates the information in the runtime structures just before the system fires Button Clicked:




The Button Clicked event is commonly used to connect to another form.

Use Repeat Business Rules for Grid to repeat ER when multiple rows are selected.

## Add Button Processing

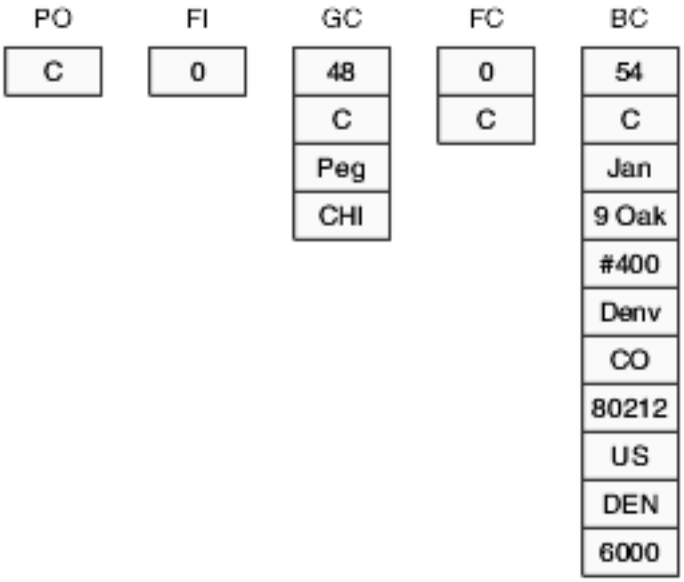
Normally, the user does not select a row before an add action, but if a row is highlighted, the system updates the GC values to reflect the selected row values. The system does not update the database just because the user clicks the row.

The engine pauses for the Button Clicked event to be processed. At this point, the runtime structures have these values:

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

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

This diagram illustrates the information that is in the runtime structures just before the system fires Button Clicked:



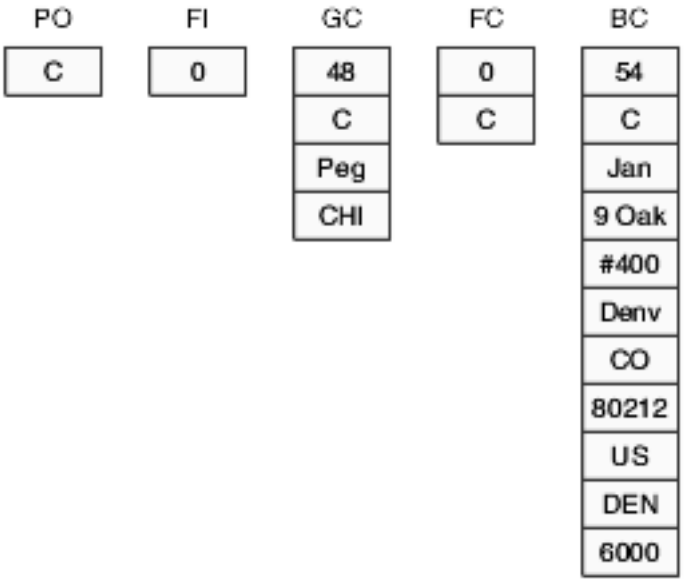

You typically use the Button Clicked event for the Add button to interconnect to another form, such as a fix/inspect or headerless detail form on which the system actually performs the add action.

## Delete Button Processing

When the user selects a grid row and clicks the Delete button, the system does not update the database immediately. The engine first fires the Button Clicked event for the Delete button. At this point, the runtime structures have these values:

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

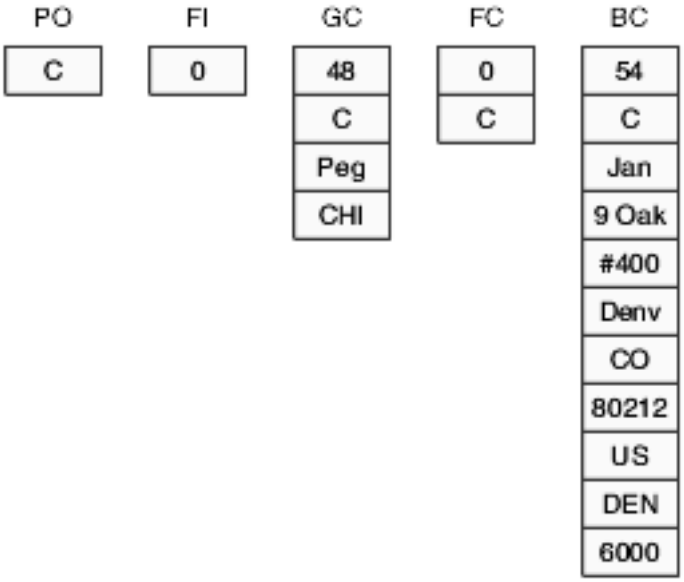
This diagram illustrates the information in the runtime structures just before the system fires Button Clicked for the Delete button:




Next, the Delete Grid Rec Verify–Before event fires.

## Delete Grid Rec Verify–Before

This diagram illustrates how the engine fires the Delete Grid Rec Verify–Before event:




Next, the system displays a pop up window for user to confirm the delete. If the delete is confirmed, the Delete Grid Rec Verify–After event fires.

## Delete Grid Rec Verify–After

In the Delete Grid Rec Verify-After event, you might want to perform custom logic to verify that the delete is valid. For example, other tables might contain dependant records that prevent this record from being deleted as long as they exist.

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

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

## Delete Grid Rec From DB–Before

At this point, the runtime structure FC is blank. The system has not yet deleted the record from the database. You can use the Suppress Delete system function in this event to prevent the system from deleting the record :

After the system processes the Delete Grid Rec From DB–Before event, it builds a `SQL DELETE` statement. Then the system deletes the current record. When user selects multiple records, all selected records are deleted.

## Delete Grid Rec From DB–After

After the records are deleted from the database, the system fires the Delete Grid Rec From DB After event You might use this event to call a business function to delete information from related tables that are not in the current BV.

## All Grid Recs Deleted From DB

After all selected records are deleted, the engine fires the All Grid Recs Deleted from DB event. At this point, FC is blank.

## Parallel Event

A parallel processing event in a new thread will occur. The parallel event processing will not interfere with existing Event Rules.

# 3 Using Event Rules Design

## Understanding Event Rules Design

You can use JD Edwards EnterpriseOne Event Rules Design to create event rule (ER) logic for forms and controls on a form. For example, you want to pass data for a selected record on a find/browse form to a fix/inspect form to revise that record. You need to create a form interconnection ER and attach it to the Select button option for the Button Clicked event.

You can create event rules that perform a large variety of tasks, including:

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

Before you create an ER, consider which control (form, button, field, grid and so on) you want to add the logic in and what event you want to add the logic for. Answer these questions to determine which event should be used:

- Is the user initializing the form?
- Is the user clicking a button?
- Is the user exiting from a field?
- Is the user changing or exiting from a row?

After you place controls on a form, you can add ERs to any of the event that the control support. Remember that a form is also a control, and you can create logic that the system automatically processes whenever a form event is fired.

You create ERs by clicking the buttons on the tool bar in JD Edwards EnterpriseOne Event Rules Design. Depending on the button that you click, a different work areas appear for creating and manipulating the ER line-by-line. Specific buttons enable you to perform these tasks:

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

- Add comments in the ER code.
- Print the ER code.

You can cut or copy an ER and paste it in the same event, form, or application or in a different event, form, or application. You can also paste ERs into other applications, such as word processing documents. This feature is useful for documenting the project.

When you paste an ER, the system resolves objects from the source as you paste them. If an object is partially resolved, the system pastes the closest matching object from the destination ER. A comment line appears above the partially-resolved line of event rules and in the status bar to indicate that the object is partially resolved. You can set paste options to display comments before and after a block of pasted ERs. Some objects cannot be resolved in the destination ER. The system disables these lines of ER and displays a comment. For example, an EndIf statement is commented out if its associated If statement is missing.

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

Use the System Function button to attach predefined system functions to events. For example, you can attach system functions to an event that perform these tasks:

- Hide or display a control.
- Display media objects.

You can attach an existing business function to an event. Business functions include these types of code:

- C code that you generate manually (source language C).
- Named Event Rules (NERs) (source language Event Rules).

You typically use business functions for these purposes:

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

# Understanding Event Rule Validation

When you save an application, the system automatically  validates all ERs in the application. If errors occur, details on the ER event, and the control and line number being executed at the time of the error are displayed in a popup window. You can also start the validation in FDA by selectingFile, Validate Event Rules.

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

# Understanding If and While Statements

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

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

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

To change a statement, double click the line.

To delete a statement, choose the line of the statement, and then click the Delete button.

# Understanding ER Consistency

You can check the consistency of events in JD Edwards EnterpriseOne Event Rule Designer. Events can be either relevant or not relevant. A *relevant* event is one that is valid and that can be executed by the associated control. A *not relevant event* is one that is not executed by the associated control because the properties of that control do not create the conditions that cause the events to execute. You can change a not relevant event to a relevant event by updating the properties of the control.

# Understanding ER Variables

An ER variable is a variable that you can use within the ER. You must assign a DD item to an ER variable. The DD item defines the type and default behavior of the variable.

Use ER variables instead of hidden fields. ER variables use fewer system resources at runtime.

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

Each ER variable is available within a scope. The scope of an ER variable determines how you can use it. Different scope options are available for interactive and batch applications. For example, you can:

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

After you create an ER variable, it appears in the available objects list in JD Edwards EnterpriseOne Event Rules Designer where you added it. Use the ER variable in ERs just as you would use any available object in the list. If you create an event level variable and do not use it in ERs, FDA automatically deletes it.

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

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

# Prerequisites

Before you complete the tasks in this section:

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

# Working with Event Rules Design

This section provides overviews of assignments and event rules design tool bar buttons, and discusses how to:

- Display event information.
- Assign a value.
- Create an If or a While statement.
- Create an ER variable.
- Attach a system function to an event.
- Attach a business function to an event.

**Note:** All event rule keywords are colored blue within Event Rules Design to easily identify.

## Understanding Assignments

Use an assignment to assign a field with a fixed value or a mathematical expression. For example, you can create an assignment that inserts a default value when the user leaves the field. You can also use an assignment to calculate a value

When you create an expression, calculate only the data items that are of the exact same numerical scale or data type. For example, do not calculate different currencies or decimal figures that represent different decimal values because the result of these calculations might compromise data integrity.

**Note:** You can use the filter capability to search for variables to use in common Event Rule functions like assignments, system functions, report interconnects, form interconnects, and business function mappings.

# Understanding Event Rules Design Tool Bar Buttons

The Event Rules Design form displays these  tool bar buttons for generating different types of statements:

- Event Information

  Displays information about event relevance.
- Assignment

  Creates an assignment or a complex expression.
- Business Function

  Attaches an existing business function.
- System Function

  Attaches an existing JD Edwards EnterpriseOne system function.
- If/While

  Creates an If/While conditional statement.
- Report Interconnect

  Establishes a connection to a batch application or report.
- Form Interconnect

  Establishes a form interconnection.
- Else

  Inserts an Else clause, which is valid only within the bounds of If and End If.
- Variable

  Creates a programmer-defined field.
- Table I/O

  Enables ER support for database access. Performs table input and output, data validations, and record retrieval.

# Displaying Event Information

To display information about the relevance of an event:

1. On Event Rules Design, select an event.
2. Click the Event Information button.

A not relevant event is displayed in dimmed italic text in the Event combo box. A message box appears that displays any information known to the system about why a not relevant event is not relevant.

# Assigning a Value

To assign a value:

1. On Event Rules Design, select an event.
2. Click the Assignment/Expression button.
3. On Assignment, select the To Object that you want to receive the assigned value.
4. Use one of these methods to determine the From/Object Literal value:
   - Select 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 (X) button to create a complex expression or advanced mathematical function using Expression Manager.

# Creating an If or a While Statement

To create an If or a While statement:

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

   Each cell in the Criteria Design grid represents a component of the criteria. When you select a cell, a list of valid options appears.
2. Select either the If or While operators.
3. Select a left operand from the list of available data items.

   Right-click to sort the available data items by name or object type. If only one type exists, the sort options are unavailable. The system groups the available data items by a variety of object types.
4. Select a logical operator comparison (is equal to, is less than, and so forth).
5. Select a right operand from the object list.
6. To assign a literal, select <Literal>.
7. To create complex If statements, you can select the And option or the Or option, and continue the logic.

**Note:** To expand or collapse all statement blocks within Event Rules Design, select Expand/Collapse All Statements in the View menu.

# Creating an ER Variable

To create an ER 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 NER.
2. Complete the variable naming field located under the Add button.

3. Click one of the Scope options (Form or Event) depending on the purpose for which you created the variable.
4. If you selected Form Scope and you want to use a grid variable, click the Grid option.
5. Click the DD visual assist to browse for DD items.
6. Select the DD item to which the variable is associated and click the Add button.

   The system automatically assigns a prefix to the variable, based on the type of scope that you choose.

## Using Event Rule Variables for Automatic Line Numbering

You can use event rules to create automatic line numbering in form grids. Automatic line numbering means that each line in the grid will have a unique number and the lines will increment automatically as grid lines are added.

To create 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_LineNumber_LNID
   ```
2. Initialize the variable on the Post Dialog is Initialized event.

   ```
   VA frm_LineNumber_LNID = 0
   ```
3. On the Grid Record is Fetched event, number the lines as each line is pulled from the database.

   ```
   If BC LineNumber > VA frm_LineNumber_LNID
   VA frm_LineNumber_LNID = BC LineNumber
   End If
   ```
4. On the Add Last Entry Row to Grid event, increment the line number and assign the new value to the next available line.

   ```
   VA frm_LineNumber_LNID = VA frm_LineNumber_LNID + 1
   GC LineNumber = VA frm_LineNumber_LNID
   ```

# Attaching a System Function to an Event

To attach a system function to an event:

1. On Event Rules Design, select an event.
2. Click the (S) button.
3. Select a category in the System Functions box.
4. Select the system function that you want to attach.
5. In the Available Objects list, select objects to pass to the system function.

# Attaching a Business Function to an Event

To attach a business function to an event:

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

   You can view a description (if one exists) for a business function by choosing Attachments from the Row menu.

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

   Range and List are not valid literals to use with business function parameters.
7. Indicate the direction of data flow between Value and Data Items, and click the OK button.

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

   - Data flows from the source to the target (right-pointing arrow).
   - Data flows from the target to the source (left-pointing arrow).
   - Data flows from the source to the target, and upon exiting the target, data flows back to the source (bi-directional arrow).
   - No data flow (slashed circle).

     If the direction of the items is hard-coded in the data structure for a business function (such as when the parameters are predetermined to be input, output, or bidirectional), then this predetermined direction appears here. You must complete the required items that appear in red. The status bar indicates the state of the flow to the target.
8. Select the Do Filtering checkbox under Transfer Filtering to transfer the end user dynamic filter criteria to the business function along with the parameter criteria.
9. Select the Include in Transaction option to include this business function for transaction processing.

   This option appears on transaction forms only.
10. Select the Asynchronously option to enable asynchronous processing.
11. Click one of these buttons to add notes:

    - Business Function Notes
    - Structure Notes
    - Parameter Notes

**Note:** To view the mapped parameters for a business function while in Event Rules Design, right click the business function and select Expand BSFN. The right click menu option will change to Collapse BSFN. To view the mapped parameters for all business functions on a specific event select Expand All BSFNs from the View menu. When expanded the Collapse All BSFNs will be activated in the View menu.

# 4 Using BrowsER

## Understanding BrowsER

You can use JD Edwards EnterpriseOne BrowsER to view event rules (ER) for interactive and batch applications.

JD Edwards EnterpriseOne BrowsER displays the structure of forms within an interactive application, or sections within a batch application. The forms or sections appear in a hierarchical structure, with events and ER for each event. You can use JD Edwards EnterpriseOne BrowsER to search and filter ER, and disable or enable ER.

You can select one of these JD Edwards EnterpriseOne BrowsER options to easily view or search for ER code:

- Expand Tree
- Expand Node
- Show Object IDs
- Hide Objects with no ER
- Filter ER Records

  Filter ER Records enables you to show or hide specific ER statements, including:

  - Assignments
  - Business functions
  - Criterion
  - Comments
  - Form interconnections
  - Options
  - System functions
- Search ER Records enables you to search for specific ER statements or text within those statements.

## Working with BrowsER

To work with JD Edwards EnterpriseOne BrowsER:

1. On Oracle's JD Edwards EnterpriseOne Object Management Workbench (OMW), select an object with ER and then click the Design button.
2. On Interactive Design, click the Design Tools tab, and then click the Browse Event Rules button.

   Alternatively, you can access JD Edwards EnterpriseOne BrowsER directly from within Oracle's JD Edwards EnterpriseOne Form Design Aid (FDA) or Oracle's JD Edwards EnterpriseOne Report Design Aid (RDA) by choosing BrowsER from the View menu.
3. On Browsing, click the plus (+) and minus (–) buttons to expand or collapse the hierarchical view of events for interactive forms or batch report sections.

Each ER appears beneath the event with which it is associated and beside a control that contains event rule logic. If it does not appear beside a control, then no event rule logic exists on that control.

4. To disable an ER line, select the line and then click Disable button.
5. To enable a disabled line, select the line and then click Enable button.

   You cannot print or modify ER from any BrowsER form.
6. To hide objects with no ER, right-click anywhere on the BrowsER window and select Hide objects with no ER from the popup menu.
7. To start a search or filter, right-click anywhere on the Browsing form and select Search or Filter ER Records from the popup menu.

# 5 Debugging Event Rules

## Understanding Debugging

Debugging is the method you use to determine the state of your program at any point of execution. Use debugging to help you solve problems and to test and confirm program execution.

Use a debugger to stop program execution so you can see the state of the program at a specific point. This enables you to view the values of input parameters, output parameters, and variables at the specified point. When program execution is stopped, you can review the code line-by-line to check such issues as flow of execution and data integrity.

You use Oracle's JD Edwards EnterpriseOne Event Rules Debugger to debug interactive applications, reports (batch applications), and table conversions.

## The Debugging Process

Use the debugging process to determine where problems occur and then fix those problems. Isolate each problem to a particular area, and then examine exactly how the program operates in that area.

If you change your program while you are debugging with the JD Edwards EnterpriseOne Event Rules Debugger, you must:

1. Exit the application.
2. Rebuild debug information.
3. Reset breakpoints.
4. Rerun the application.

Features available in the event rule debugger are listed and described in this table:

| Feature | Description |
|---|---|
| Go | Command that resumes program execution after a breakpoint is reached. |
| Breakpoint | Command that tell the debugger to stop when a particular line is reached. You can set breakpoints on lines of code where you want to start debugging. |
| Delete Breakpoint | Command that removes all breakpoints that you currently have set. |
| Step, Step Over | Command that executes the current line of code. Step lets you run the program one line at a time. You can use this feature to determine the results of every line of code as it is executed. |
| Step Into | Command used when the current line of code contains a function call. The debugger will step into the function so that it can be debugged line by line. When the function is complete, the debugger returns to the next line of code after the function call in the calling routine. Step Into can be used to debug a second application that is called from within an application. |

| Feature | Description |
|---|---|
| | |
| Disconnect | Command that disconnects the debugger from the current application. The application continues to run as if the debugger had not been started. |

# Debugging Strategies

You can use several strategies to make debugging faster and easier. Begin by observing the nature of the problem.

## Is the Program Ending Unexpectedly?

If the program is ending unexpectedly, the cause is likely an unhandled exception. It is an easy problem to track down if it is happening in the same place: simply set breakpoints at strategic points throughout the code and run the program until you find the problem.

If other objects are missing, termination is more abrupt. Remember to transfer all Media Object (also called Generic Text) objects correctly. If an application has a Row exit to an application that does not exist, an unhandled exception in the program occurs immediately.

Termination of the program is more abrupt and less helpful when other kinds of objects are missing. You must review all of the pieces of your application to verify that they are all present and correctly built. A common error is to overlook media objects. If you cannot enter your program at all, a missing object is most likely the problem.

## Is the Output of the Program Incorrect?

Incorrect program output typically indicates a flaw within the logic of the code. To help find the error:

- Set a breakpoint in the code prior to the point where the bad output is produced.
- Step through the ER line by line, while monitoring the values of relevant ER variables.

  At some point, a variable will probably take on an erroneous value that subsequently produces incorrect output.
- If that point occurs before your breakpoint, set another breakpoint earlier in the code and restart the application.
- Continue this process until you find the statement that is causing the wrong value to be assigned to the variable.

## Where Else Could the Problem Be Coming From?

Spend some time thinking about where the source of the problem might be. If you don't know which ER event is causing an error, try to isolate it. For example, you might be able to temporarily disable the ER one event at a time to see if the error still happens. You can try to repeat the processing of a single event by doing unnatural actions in the GUI, like toggling up and down between grid rows to force the execution of the Row Is Exited event. There are no predefined

debugging strategies that will work in any given situation. Be creative and be persistent, until you narrow down the problem to its source.

# Debug Logs

You can output to a file a log of SQL statements and events by changing the line in your jde.ini file under **[DEBUG]** from **Output = NONE** to **Output = FILE**, as in this sample. This is a useful debugging tool when you have narrowed a problem to a specific issue involving the JDEDB APIs.

```
[DEBUG]
TAMMULTIUSERON=0
Output=FILE
ServerLog=0
LEVEL=BSFN,EVENTS
DebugFile=c:\jdedebug.log
JobFile=c:\jde.log
Frequency=10000
RepTrace=0
```

The jdedebug.log file can become very large when you set the Output=DEBUG flag. It is very difficult to work through all the log messages and correlate them to what was happening in the application. One way to aid in this process is to set breakpoints in the ER, and each time the application stops at a breakpoint, make a separate copy of the jdedebug.log file. Label each copy, and when you are done, you will have narrowed down the coverage of each portion of the jdedebug.log file a bit.

# Debugging Event Rules

This section provides an overview of the JD Edwards EnterpriseOne Event Rules Debugger and discusses how to:

- Debug an application with the JD Edwards EnterpriseOne Event Rules Debugger.
- Inspect or modify a variable.

## Understanding the Event Rules Debugger

The JD Edwards EnterpriseOne Event Rules Debugger provides the essential debugging tools (such as breakpoints, step commands, and variable inspection) that you need to debug JD Edwards EnterpriseOne interactive and batch applications. You can debug both NERs and table ER. The generated debug information for an application includes NER and table ER information for that application.

Setting up and using JD Edwards EnterpriseOne Event Rules Debugger involves these steps:

1. Launch the ER Debugger.
2. Load into the Debugger the applications to debug.
3. Set any desired breakpoints.
4. Launch the application, report, or table conversion.

Step 4 may be done at any point before, during or after steps 1, 2, or 3.

Step 2 takes a few minutes. The Debugger must read all the specs for the application, and then translate them into a usable format (Debugging Information Archive, or DIA). For large applications, this is slow. Therefore, once you load an

application, you don't want to have to reload it again any time soon (unless you modify the application, of course). For this reason, the ER Debugger provides a Deactivate feature. Deactivating an application prevents any debugging from occurring on that application. But when you are ready to debug the application later, you won't have to rebuild the DIA.

Any event on which you want the debugger to stop must have at least one line of ER code. You cannot set a breakpoint on a comment. When you debug an application and encounter a point at which the interactive or batch application fails, you can view the code that the application is currently executing, and inspect the live values of variables used in the code.

By observing specific variables while the program runs, you can isolate where the program begins to fail and what exactly it is doing. For example, if a counter is supposed to increment by 1, but you observe it incrementing by random numbers, you know there is a problem with the number or variable you are adding to the counter.

The JD Edwards EnterpriseOne Event Rules Debugger is a standalone tools program. Its screen interface consists of these main components:

- Object Browse window.
- Object Tree (those applications currently loaded in Debugger).
- Event Rules window (ER listing).
- Breakpoint Manager.
- Variables List and Variable Watch window.

All windows except the Event Rules window are dockable to any side of the main application. You can click and drag a window to dock it. When you close the JD Edwards EnterpriseOne Event Rules Debugger, it saves your docking settings until the next time you run it. Most of these main windows may also be closed by clicking the large toolbar buttons which control them, or by clicking the 'X' in the top border of the window.

## Object Browse Window

The Object Browse window is used to locate applications to load into the debugger. It provides three tab pages, one for interactive applications, one for UBEs, and one for TCs. You can find the object you want to debug in one of these lists, and then select it in order to bring it into the debugger.

Once you are done loading objects into the Debugger, you may want to close, or hide, the Object Browse window, because it takes up screen space that could be more usefully dedicated to one of the other Debugger windows. To hide this window, toggle the Object Browse Window button in the toolbar, or use the View menu.

## Object Tree

The Object Tree view lists applications that have debug information built and are available for debugging. You can navigate through a tree structure to a specific event and open an Event Rules window for that event. If one of the objects is a power form with subforms, the subforms are listed under the power form. Form IDs appear next to the form name.

## Event Rules Window

Each Event Rules window displays the ER for one event. The event name and path, along with an abbreviated description, appear in the title bar for each Event Rules window. The Event Rules window shows the line in the ER that is currently being executed when the runtime engine is stopped on a line break.

The left side of an Event Rule window displays icons that describe the state of a line in the ER. States include breakpoint, disabled, or current line of execution.

You can use the Event Rules window to set and remove breakpoints. You can use any of these methods:

- Double-click the line in ER.
- Right-click a line and select Insert Breakpoint or New Breakpoint from the pop-up menu.
- Select a line and press F9.
- Single click in the margin to the left of the ER text.

You cannot set a breakpoint on a comment line. The breakpoint automatically goes to the first code line after the comment. You cannot set a breakpoint on a data structure member displayed in the ER listing. The breakpoint will automatically go to the function call above it that "owns" the data structure.

The ER window is also useful to show the values of ER variables during a debugging session. The value of a variable in the ER listing can be instantly displayed by moving the mouse cursor over the variable. Its value will appear in a Tool Tip window when the ER Engine is paused (when stopped on a breakpoint or when stepping through code in the debugger).

Variables spelled differently in the ER listing and the variable list will not display.

## Variable Tree and Watch Window

When the program is halted at a breakpoint, you can examine the state of your runtime structures and evaluate ER variables using the Variable Tree and Variable Watch windows.

The Variable Tree and Watch window consists of two panes. The left pane is the Variable Tree pane. It contains a tree structure that lists the variable types as parent nodes and the variables of each variable type as child nodes. The variables displayed are those that are in scope of the currently displayed event in the Event Rule window. The right pane is the Watch pane. It displays variables you select and their most recently known values. Each variable is identified by its category, name, and the Id of the form, if any, to which it belongs. You can add a variable to the Watch pane by double-clicking the desired variable in the Variable Tree, or you can right click on the variable in the Variable Tree and select "Watch Variable" to add it to the Watch list.

You can change the value of variables while you are debugging an application, in order to better understand what effect that might have on subsequent ER execution. This is a powerful feature, which can also mess up your program execution and create unwanted side effects. Use it carefully. To change the value of a variable, the application must be stopped at a line of ER. Double-click the variable in the Watch pane and enter a different value. If the ER engine running the application accepts the new value, the new value appears in the Watch pane. If you enter an inappropriate value (for example, you change a numeric value to an alpha value) the new value is not set and the value is not changed. In any case, the Debugger displays the actual, resulting variable value.

These special values are displayed for variables:

- blank

  The value for the variable contains only blanks. This value applies to string and character types only.

- null

  The variable has no value, or a null or empty value.

- unknown

  The value for the variable could not be obtained from the runtime engine. When adding a variable to the watch list, the initial value is always unknown. The value is also unknown when the applications are not running, or when the variable is out of scope.

  > **Note:** Variable inspection and modification is not available for debugging NERs and table ER.

## Breakpoint Manager

You use breakpoints to define where or when to halt the execution of a program. The Breakpoint Manager tracks the breakpoints that are set and the location of those breakpoints in an application. When you set a new breakpoint, the system creates an entry in the Breakpoint Manager. This entry contains the application name, form name, event name, ER line number, and the breakpoint conditions, if any.

Right-click within Breakpoint Manager to perform these operations:

- Delete a breakpoint.
- Delete all breakpoints.
- Display the ER window in which the breakpoint is set.
- Access breakpoint properties.
- Enable or disable a breakpoint.

You can also double-click an entry in Breakpoint Manager to open the Event Rule window in which the breakpoint is set.

## Breakpoint State Indicators

Normal breakpoints are indicated by a red circle.

Breakpoint conditions are indicated by a question mark inside the red circle.

A disabled breakpoint is indicated by a hollow circle in the ER listing.

## Breakpoint with Condition

Formerly, ER breakpoints existed with no apparent properties. Breakpoints now have Condition and Hit Count properties which may be set or examined in the Breakpoint Properties dialog.

To set a breakpoint condition in the form of a logical expression:

1. In the Breakpoint Properties dialog click the Condition button.
2. The Breakpoint Condition dialog will now display. In this window you may enter the condition in the available field or click the Help button. The Help button brings up a dialog that documents the syntax for breakpoint conditions to assist in creating a condition.

   > **Note:** For example one might write a condition using the format of "VariableName = LiteralValue". The resulting condition might be PO cSelfServiceMode='1'.

3. After entering a condition, the developer may use the Validate Condition button to validate the expression.

   > **Note:** Checking the Condition checkbox in the Breakpoint Condition dialog enables the Condition. If the Condition checkbox is not checked then the condition is saved for possible later use, but is not effective.

## Avoiding Problems with Breakpoint Conditions

To avoid unnecessary problems it is best to:

- Validate breakpoint conditions as you create them.
- Spell variable names exactly as they appear in the variable list, including spaces and special characters where present.

It is possible to set an invalid condition on a breakpoint. There may even be times when you would do this intentionally. Invalid conditions fail when evaluated. It is good practice to utilize the Validate Condition button to validate conditions as you create them.

Misspelled variable names are invalid, of course. The official spelling for all variables is shown in the Debugger's variable list.

Out-of-scope variables are invalid, depending on the context of the breakpoint in which the condition is set.

Any other syntax violations make an expression invalid.

The Debugger does not validate date and number formats. This validation is done at runtime, and may depend on certain User Options.

## Breakpoint with Hit Count Condition

The second Breakpoint property is the Hit Count condition. The Hit Count condition specifies a certain number of times the breakpoint must be reached before the debugger will stop on it.

The Hit Count dialog is used by the developer to specify a Hit Count condition. It also shows the current hit count, that is, the number of times the breakpoint has been reached since the hit count condition was created.

When both an expression Condition and a Hit Count condition are set on a single breakpoint, the debugger stops on the breakpoint if either condition is met.

## Disabled Breakpoint

A breakpoint may be disabled. A disabled breakpoint retains all its conditions and other properties, but the debugger does not stop on a disabled breakpoint.

## Invalid Breakpoints

Invalid breakpoint conditions are logged to jdedebug.log each time the breakpoint having the invalid condition is reached.

## Search Combo Box

You can use the Search combo box on the tool bar to search for ER text. Enter the text that you want to find in the Search combo box and then press either Enter or F3. If the system locates the search text in your ER text, it highlights the text. If you press Enter or F3 again, the next occurrence of your search text is located and highlighted.

The search control accommodates regular expression searches. A regular expression search uses special characters to match text. For example, `^If:` will find every line that starts with If and `If$:` will find every line that ends with If.

The special characters that you can use for advanced searches are described in this table:

| Character | Description |
|---|---|
| ^ | The caret (^) indicates the beginning of a line. For example, the expression ^A matches an A only at the beginning of a line. |
| ^ | The caret (^) immediately after the left bracket ( [ ) is used to exclude any remaining characters within brackets from matching the target string. For example, the expression [^0-9] indicates that the target character should not be a digit. |

| Character | Description |
|---|---|
| $ | The dollar sign ($) matches the end of a line. For example, the expression abc$ will match the substring abc only if it is at the end of a line. |
| \| | The alternation character (\|) enables the expression on either side of it to match the target string. For example, the expression a\|b will match a as well as b. |
| . | The dot (.) matches any character. |
| * | The asterisk (*) indicates that the character to the left of the asterisk in the expression should match 0 or more times. |
| + | The plus (+) is similar to the asterisk except that at least one match of the character should occur to the left of the + sign in the expression. |
| ? | The question mark (?) matches the character to its left 0 or 1 times. |
| () | The parentheses affect the order of pattern evaluation and serve as a tagged expression that you can use to replace a matched substring with another expression. |
| [ ] | Brackets that enclose a set of characters indicate that any of the enclosed characters can match the target character. |

# Debugging an Application with the Event Rules Debugger

To debug an application with the JD Edwards EnterpriseOne Event Rules Debugger:

1. From the Cross Application Development Tools menu (GH902), select Debug Application.
2. Select the object that you want to debug.
3. Select a form (for interactive applications) or section (for batch applications) and an event to view.
4. Select the ER line on which you want to set a breakpoint.
5. Select Debug, Breakpoint.

   A red dot appears on the line, indicating the breakpoint.o You can remove the breakpoint by choosing Debug, Breakpointagain. The Breakpoint command is a toggle, and you can also toggle the value using the Breakpoint toolbar button.
6. Run the application.

   As your application encounters a breakpoint, the application will pause, and the focus will switch to the Event Rules Debugger.

   When execution stops at a breakpoint, you can use the variables view to inspect and modify the values of runtime structures.

7. From the Debug menu, select one of these options:
    - Go
    - Disconnect
    - Step Over
    - Step Into

# 6 Glossary

## business function

A named set of user-created, reusable business rules and logs that can be called through event rules. Business functions can run a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the application programming interfaces (APIs) that enable them to be called from a form, a database trigger, or a non-JD Edwards EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.

## business function event rule

See named event rule (NER).

## business view

A means for selecting specific columns from one or more JD Edwards EnterpriseOne application tables whose data is used in an application or report. A business view does not select specific rows, nor does it contain any actual data. It is strictly a view through which you can manipulate data.

## embedded event rule

An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with the business function event rule.

## event rule

A logic statement that instructs the system to perform one or more operations based on an activity that can occur in a specific application, such as entering a form or exiting a field.

# fast path

A command prompt that enables the user to move quickly among menus and applications by using specific commands.

# jde.ini

A JD Edwards EnterpriseOne file (or member for iSeries) that provides the runtime settings required for JD Edwards EnterpriseOne initialization. Specific versions of the file or member must reside on every machine running JD Edwards EnterpriseOne. This includes workstations and servers.

# jde.log

The main diagnostic log file of JD Edwards EnterpriseOne. This file is always located in the root directory on the primary drive and contains status and error messages from the startup and operation of JD Edwards EnterpriseOne.

# named event rule (NER)

Encapsulated, reusable business logic created using event rules, rather that C programming. NERs are also called business function event rules. NERs can be reused in multiple places by multiple programs. This modularity lends itself to streamlining, reusability of code, and less work.

# subscriber table

Table F98DRSUB, which is stored on the publisher server with the F98DRPUB table and identifies all of the subscriber machines for each published table.

# table conversion

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

## table event rules

Logic that is attached to database triggers that runs whenever the action specified by the trigger occurs against the table. Although JD Edwards EnterpriseOne enables event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

## workbench

A program that enables users to access a group of related programs from a single entry point. Typically, the programs that you access from a workbench are used to complete a large business process. For example, you use the JD Edwards EnterpriseOne Payroll Cycle Workbench (P07210) to access all of the programs that the system uses to process payroll, print payments, create payroll reports, create journal entries, and update payroll history. Examples of JD Edwards EnterpriseOne workbenches include Service Management Workbench (P90CD020), Line Scheduling Workbench (P3153), Planning Workbench (P13700), Auditor's Workbench (P09E115), and Payroll Cycle Workbench.

# Index

## R



## S



## T



## U



## V



## W