

JD Edwards EnterpriseOne Tools

Form Design Aid Guide

Release 9.2

E53552-05

October 2017

Describes how to use the Form Design Aid in EnterpriseOne to create or modify EnterpriseOne interactive applications.

Copyright © 2014, 2017, Oracle and/or its affiliates. All rights reserved.

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 installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 Xeon 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, Opteron, the AMD logo, and the AMD Opteron 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	xix
Audience.....	xix
Documentation Accessibility	xix
Related Information	xix
Conventions	xx
1 Introduction to JD Edwards EnterpriseOne Tools Form Design Aid	
1.1 Development Tools Form Design Aid Overview.....	1-1
1.2 Development Tools Form Design Aid Implementation.....	1-1
1.2.1 JD Edwards EnterpriseOne Tools Form Design Aid Implementation Steps	1-2
2 Working with Forms	
2.1 Understanding Forms	2-1
2.2 Understanding Form Interconnections.....	2-2
2.3 Configuring Forms at Design Time.....	2-3
2.4 Creating Forms.....	2-8
2.4.1 Understanding Form Creation.....	2-8
2.4.1.1 Recommended FDA Configuration.....	2-9
2.4.2 Creating a Form	2-10
2.4.3 Creating a Modal Form Interconnection	2-10
2.4.4 Creating a Dynamic Modal Form Interconnection.....	2-11
2.4.5 Creating a Modeless Form Interconnection.....	2-11
2.4.6 Creating a Pop-up Form Interconnection (Release 9.2.1).....	2-12
2.5 Working with the Accessibility Violation Check Feature	2-13
2.5.1 Understanding Accessibility Violation Check Feature	2-13
2.5.2 Enabling the Accessibility Violation Check Feature.....	2-13
2.5.3 Identifying Empty Tooltips Using Accessibility Violation Check Feature	2-14
2.6 Working with Quick Form	2-15
2.6.1 Prerequisite	2-16
2.6.2 Using Quick Form	2-16
3 Working with Form Controls	
3.1 Understanding Form Controls.....	3-1
3.2 Understanding Form Control Design-Time Considerations.....	3-4
3.3 Understanding Chart Control.....	3-24

3.3.1	Using the Chart Control.....	3-24
3.3.2	Using XML Graph Access.....	3-25
3.3.3	System Functions for Charting	3-25
3.3.4	C Code System Functions	3-27
3.3.5	Example Data XML	3-27
3.3.6	Example Graph XMLs.....	3-29
3.3.7	Example Graph DTD.....	3-37
3.4	Attaching Data Items to a Control.....	3-40
3.4.1	Understanding the Relationship between Data Dictionary Items and Controls....	3-40
3.4.2	Attaching a Data Item to a Control.....	3-41
3.4.3	Overriding a Default Data Dictionary Trigger.....	3-41
3.5	Associating a Data Item Description with a Field.....	3-42
3.5.1	Understanding the Relationship between Data Item Descriptions and Fields	3-42
3.5.2	Displaying the Title of a Data Item Associated with a Field	3-42
3.6	Grouping Controls.....	3-42
3.7	Setting the Tab Sequence of Controls on a Form	3-43
3.7.1	Understanding Tab Sequences	3-43
3.7.2	Changing the Tab Sequence on a Form	3-43
3.8	JD Edwards EnterpriseOne FDA Compare	3-44

4 Working with Transaction Processing

4.1	Understanding Transaction Processing.....	4-1
4.1.1	Commits and Rollbacks	4-2
4.1.1.1	Commit.....	4-2
4.1.1.2	Rollback.....	4-2
4.1.2	Transaction Processing	4-2
4.1.2.1	Data Interdependence.....	4-3
4.1.2.2	Transaction Boundaries	4-3
4.1.2.3	Interactive Application Transaction Processing Scenarios.....	4-3
4.1.2.4	Transaction Processing and Business Functions.....	4-5
4.1.2.5	Transaction Processing in Remote Business Functions	4-5
4.1.3	Transaction Processing Availability.....	4-6
4.2	Implementing Transaction Processing.....	4-6
4.2.1	Forms Used to Work with Transaction Processing in Interactive Applications	4-7
4.2.2	Defining Transaction Processing for a Form	4-7
4.2.3	Extending a Transaction Boundary.....	4-7
4.2.3.1	Extending a Transaction Boundary between Forms	4-7
4.2.3.2	Extending a Transaction Boundary by Using Business Functions.....	4-8
4.2.3.3	Extending a Transaction Boundary by Using Table I/O.....	4-8
4.2.3.4	Defining Transaction Processing for a Report.....	4-9

5 Understanding Find/Browse Forms

5.1	Find/Browse Forms.....	5-1
5.2	Find/Browse Events.....	5-1
5.3	Find/Browse Runtime Processing	5-1
5.3.1	Dialog Initialization.....	5-2
5.3.2	Find Button	5-2

5.3.3	Select Button	5-3
5.3.4	Close Button	5-3
5.3.5	Dialog Close.....	5-3

6 Understanding Fix/Inspect Forms

6.1	Fix/Inspect Forms.....	6-1
6.2	Fix/Inspect Events.....	6-1
6.3	Fix/Inspect Runtime Processing.....	6-1
6.3.1	Dialog Initialization.....	6-1
6.3.2	Dialog Clear.....	6-2
6.3.3	Data Retrieval.....	6-3
6.3.4	OK Button	6-3
6.3.5	Cancel Button	6-7
6.3.6	Dialog Close.....	6-7

7 Understanding Header Detail Forms

7.1	Header Detail Forms	7-1
7.2	Header Detail Design-Time Considerations	7-1
7.3	Header Detail Events.....	7-1
7.4	Header Detail Runtime Processing	7-2
7.4.1	Dialog Initialization.....	7-2
7.4.2	Dialog Clear	7-3
7.4.3	Data Retrieval.....	7-4
7.4.4	OK Button	7-4
7.4.5	Cancel Button	7-8
7.4.6	Dialog Close.....	7-8

8 Understanding Headerless Detail Forms

8.1	Headerless Detail Forms	8-1
8.2	Headerless Detail Design-Time Considerations.....	8-1
8.3	Headerless Detail Events	8-1
8.4	Headerless Detail Runtime Processing.....	8-2
8.4.1	Dialog Initialization.....	8-2
8.4.2	Dialog Clear	8-3
8.4.3	Find Button	8-4
8.4.4	OK Button	8-4
8.4.5	Delete Button	8-8
8.4.6	Cancel Button	8-8
8.4.7	Dialog Close.....	8-8

9 Understanding Message Forms

9.1	Message Forms	9-1
9.1.1	Hover Events	9-1
9.1.2	Hover System Functions.....	9-2
9.2	Message Form Design-Time Considerations	9-3

9.3	Understanding Message Form Events.....	9-3
9.4	Message Form Runtime Processing	9-3
9.4.1	Dialog Initialization.....	9-3
9.4.2	Dialog Close.....	9-4

10 Understanding Parent/Child Browse Forms

10.1	Parent/Child Browse Forms	10-1
10.2	Parent/Child Browse Events	10-1
10.3	Parent/Child Browse Runtime Processing	10-1
10.3.1	Dialog Initialization.....	10-1
10.3.2	Find Button	10-2
10.3.3	Select Button	10-2
10.3.4	Close Button	10-2
10.3.5	Dialog Close.....	10-3

11 Understanding Portlet Forms

11.1	Portlet Design Considerations	11-1
11.2	Portlet Types	11-1
11.2.1	Portlets that are Alerts.....	11-1
11.2.2	Portlets that are Menus	11-2
11.2.3	Portlets that are Shortcuts.....	11-2
11.3	Understanding Portlet Forms.....	11-2
11.3.1	Portlet Form Features.....	11-2
11.3.2	Portlet Personalization	11-3
11.3.3	Portlet Form Events	11-3
11.3.4	Edit Portlet Form Design-Time Considerations.....	11-4
11.4	Generating Portlets (Prior to release 9.2.2).....	11-4
11.4.1	Prerequisites	11-4
11.4.2	Understanding Portlet Generation.....	11-4
11.4.3	Deploying an FDA-Created Portlet.....	11-5
11.5	Updating an FDA-Created Portlet after Initial Installation (Prior to release 9.2.2).....	11-5
11.5.1	Understanding Portlet Updates.....	11-5
11.5.2	Adding a New Portlet Application	11-5
11.5.3	Deleting an Existing Portlet Application.....	11-6

12 Understanding Power Browse Forms

12.1	Power Browse Forms.....	12-1
12.2	Power Browse Form Hierarchical Structures.....	12-2
12.2.1	Examples of the Logic Flow of Power Forms	12-2
12.3	Power Browse Form Design-Time Considerations.....	12-4
12.4	Power Browse Events	12-4
12.5	Power Browse Runtime Processing.....	12-5
12.5.1	Dialog Initialization.....	12-5
12.5.2	Find Button	12-6
12.5.3	Select Button	12-6
12.5.4	Close Button	12-6

12.5.5	Dialog Close.....	12-6
12.6	Transaction Boundaries for Power Browse Forms and Subforms.....	12-6
13	Understanding Power Edit Forms	
13.1	Power Edit Forms	13-1
13.2	Power Edit Form Design-Time Considerations.....	13-1
13.3	Power Edit Events.....	13-2
13.4	Power Edit Form Runtime Processing.....	13-2
13.4.1	Dialog Initialization.....	13-2
13.4.2	Dialog Clear.....	13-5
13.4.3	OK Button	13-5
13.4.4	Cancel Button	13-9
13.4.5	Dialog Close.....	13-9
14	Understanding Search & Select Forms	
14.1	Search & Select Forms	14-1
14.2	Search & Select Events.....	14-1
14.3	Search & Select Runtime Processing	14-1
14.3.1	Dialog Initialization.....	14-2
14.3.2	Find Button	14-2
14.3.3	Select Button	14-2
14.3.4	Close Button	14-2
14.3.5	Dialog Close.....	14-3
15	Understanding Wizard Forms	
15.1	Wizard Forms	15-1
16	Understanding External Forms (Release 9.2.1)	
16.1	External Forms Overview	16-1
16.2	External Form Properties	16-2
16.2.1	External Application Type Property	16-2
16.2.2	External Application Property	16-2
16.2.2.1	JavaScript	16-2
16.2.2.2	ADF.....	16-2
16.2.2.3	Composed EnterpriseOne Page.....	16-3
16.2.3	Entry Point Property	16-3
16.3	Data Structures.....	16-3
17	Understanding Calendar Controls	
17.1	Calendar Controls	17-1
17.2	Calendar Control Design-Time Considerations	17-2
17.3	Calendar Control Events.....	17-2
17.3.1	Load Calendar Activity.....	17-2
17.3.2	Drill Into Calendar Activity	17-3
17.3.3	Drill Into Time Span	17-3

17.3.4	Add Activity Button Clicked and Post Activity Button Clicked	17-3
17.4	Calendar Control Runtime Processing	17-3
17.4.1	Initialize the Control.....	17-3
17.4.2	Add a Calendar Activity.....	17-4
17.4.3	Refresh the Control.....	17-4
17.5	Calendar Control System Functions	17-4
	Add Calendar Activity	17-6
	Delete Calendar Activity.....	17-9
	Modify Calendar Activity.....	17-10
	Select Calendar View.....	17-13

18 Understanding Check Box Controls

18.1	Check Box Controls	18-1
18.2	Check Box Control Design-Time Considerations	18-1
18.3	Check Box Events.....	18-1

19 Understanding Combo Box Controls

19.1	Understanding Combo Box Controls.....	19-1
19.2	Loading Combo Box Controls.....	19-2
19.2.1	Loading Combo Box Fundamentals	19-2
19.2.2	Loading a Combo Box from a UDC	19-2
19.2.3	Loading a Combo Box from Cache	19-3
19.2.4	Loading a Combo Box with the Add Item System Function.....	19-4
19.3	Combo Box Control Design-Time Considerations.....	19-4
19.4	Combo Box Control Events	19-4
19.5	Combo Box Control Runtime Processing	19-5
19.5.1	Control Initialization	19-5
19.5.2	Control Validation.....	19-7
19.5.3	Load from Cache.....	19-7
19.5.4	Database Commit	19-8
19.5.5	System Functions	19-8
19.5.6	Import into Grid.....	19-9
19.6	Combo Box Control System Functions	19-9
	Add Item	19-10
	Get Description.....	19-11
	Get Index of Key.....	19-12
	Get Item at Index.....	19-13
	Get Item Count	19-14
	Get Key at Index.....	19-15
	Load from Cache.....	19-16
	Remove Item by Index	19-18
	Remove Item by Key	19-19
	Select Item	19-20
	Embedded Combo Box System Functions	19-21

Add Item	19-22
Get Description.....	19-23
Get Index of Key.....	19-24
Get Item at Index.....	19-25
Get Item Count.....	19-26
Get Key at Index.....	19-27
Load from Cache	19-28
Remove Item by Index	19-30
Remove Item by Key	19-31
Select Item	19-32

20 Understanding Edit Controls

20.1 Edit Controls.....	20-1
20.2 Edit Control Events.....	20-2
20.3 Edit Control Runtime Processing	20-2
20.3.1 Control is Entered.....	20-2
20.3.2 Control is Exited	20-3
20.4 Edit Control System Functions	20-3
Set Edit Control Color	20-4
Set Edit Control Font	20-5

21 Understanding Grid Controls

21.1 Grid Controls	21-1
21.2 Grid Control Design-Time Considerations	21-2
21.2.1 Designing the Grid	21-2
21.2.2 Adding Columns to the Grid Control.....	21-3
21.2.3 Displaying Grid Data as Icons	21-3
21.2.3.1 Icon Display States	21-4
21.2.3.2 Associating Icons to Data Values	21-4
21.2.3.3 Tooltips on Icons.....	21-4
21.2.3.4 Implementing Icons on Grids	21-5
21.2.4 Setting Property Values for the Grid Control.....	21-5
21.2.4.1 Grid Control Display	21-5
21.2.4.2 Loading and Processing Behavior.....	21-7
21.2.4.3 Data Entry Behavior	21-8
21.2.5 Showing Multiple Currencies per Column.....	21-9
21.2.6 Adding Aggregation to Grid Column	21-10
21.3 Grid Control Events.....	21-10
21.4 Grid Control Runtime Processing	21-12
21.4.1 How Runtime Processes the Grid Control.....	21-12
21.4.2 Impact of Interactivity Levels	21-22
21.5 Grid Control System Functions	21-23
Change Row Selection.....	21-27
Clear Grid Buffer.....	21-28

Clear Grid Cell Error	21-29
Clear QBE Column	21-30
Clear Selection	21-31
Clear Sequencing.....	21-32
Copy Grid Row to Grid Buffer.....	21-33
Delete Grid Row.....	21-34
Disable Grid	21-35
Display Customized Grid Option	21-36
Display Export to Excel Option	21-37
Display Export to Word Option.....	21-38
Display Import from Excel Option.....	21-39
Enable Grid	21-40
Get Grid Row	21-41
Get Max Grid Rows	21-42
Get Next Selected Row	21-43
Get Selected Grid Row Count	21-44
Get Selected Grid Row Number	21-45
Hide Grid Column.....	21-46
Hide Grid Row	21-47
Insert Grid Buffer Row	21-48
Insert Grid Row	21-49
Set Data Dictionary Item.....	21-50
Set Data Dictionary Item Overrides	21-51
Set Grid Cell Error	21-52
Set Grid Cell Icon	21-53
Set Grid Cell Icon Visibility	21-54
Set Grid Color	21-55
Set Grid Column Heading	21-56
Set Grid Font	21-57
Set Grid Row Bitmap	21-58
Set Grid Row Format.....	21-59
Set Lower Limit	21-60
Set QBE Column Compare Style.....	21-62
Set Selection	21-63
Set Selection Group.....	21-64
Set Selection Append Flag	21-66
Set Sequencing.....	21-67
Show Grid Column.....	21-68
Show Grid Row	21-69
Suppress Grid Line	21-70
Update Grid Buffer Row.....	21-71
Was Grid Cell Value Entered	21-73

22 Understanding Hot Keys

22.1	System-Defined Push Button Hot Keys.....	22-1
22.2	System-Defined Toolbar Button Hot Keys.....	22-1
22.3	Application-Defined Hot Keys	22-1
22.4	Defining a Hot Key in Your Application.....	22-2

23 Understanding Image Controls

23.1	Image Controls	23-1
23.2	Image Control Design-Time Considerations	23-1

24 Understanding Media Object Controls

24.1	Media Object Controls.....	24-1
24.2	Media Object Control Design-Time Considerations.....	24-1
24.3	Media Object System Functions.....	24-2
	Access Media Object.....	24-3
	Manage Media object	24-4
	Activate Item.....	24-5
	Clear Characterization Cache.....	24-6
	Delete Item	24-7
	Disable Characterization Cache	24-8
	Get OLE Item	24-9
	Insert OLE Object.....	24-10
	Insert Text.....	24-11
	Insert URL	24-12
	Hide the Viewer Icon Panel.....	24-13
	Lock the Viewer Splitter Bar.....	24-14
	Set Characterization Cache	24-15
	Set Cursor Position	24-16
	Set Grid Text Indicator	24-17
	Set Text Color	24-18

25 Understanding Parent Child Controls

25.1	Parent Child Controls.....	25-1
25.2	Tree Nodes	25-2
25.3	Lean Manufacturing	25-2
25.4	Parent Child Control Design-Time Considerations.....	25-2
25.4.1	Parent Child Control Properties	25-3
25.4.2	Parent Child Control and Power Forms.....	25-3
25.4.3	Lean Manufacturing Properties.....	25-4
25.5	Parent Child Control Events	25-4
25.5.1	Selecting Tree Nodes	25-5
25.5.2	Performing Drag-and-Drop or Copy/Cut/Paste	25-5
25.5.3	Expanding and Collapsing Nodes	25-7

25.5.3.1	Example: Using the Tree Node is Expanded Event.....	25-7
25.5.4	Clicking Bitmaps.....	25-9
25.6	Parent Child Control System Functions	25-9
	Add Action.....	25-11
	Attach Path To Segment.....	25-12
	Change Row Selection.....	25-13
	Clear Grid Buffer.....	25-14
	Clear Grid Cell Error	25-15
	Clear QBE Column	25-16
	Contact Tree Node	25-17
	Copy Grid Row To Grid Buffer.....	25-18
	Delete All Actions	25-19
	Delete All Tree Nodes	25-20
	Delete Grid Row.....	25-21
	Disable Grid	25-22
	Enable Grid	25-23
	Expand Tree Node	25-24
	Get Grid Row.....	25-25
	Get Max Grid Rows	25-26
	Get Next Selected Row	25-27
	Get Node ID.....	25-28
	Get Node Level.....	25-29
	Get Related Node ID	25-30
	Get Row Number	25-31
	Get Selected Context Action.....	25-32
	Get Selected Grid Row Count	25-33
	Get Selected Grid Row Number	25-34
	Get Tree Node Handle	25-35
	Hide Grid Column	25-36
	Insert Grid Buffer Row	25-37
	Insert Grid Buffer Row By Node ID.....	25-38
	Set Action	25-40
	Set Data Dictionary Item.....	25-41
	Set Data Dictionary Overrides	25-42
	Set Drag Cursor	25-43
	Set Grid Cell Error	25-44
	Set Grid Color	25-45
	Set Grid Column Heading	25-46
	Set Grid Font	25-47
	Set Grid Row Bitmap	25-48
	Set QBE Column Compare Style.....	25-49
	Set Tree Bitmap Scheme	25-50

Set Tree Node Bitmap.....	25-51
Set Tree Node Bold	25-52
Set Tree Node Clickable Bitmap	25-53
Set Tree Node Handle	25-54
Set Tree Root Node ID.....	25-55
Show Grid Column.....	25-56
Show N Levels.....	25-57
Suppress Fetch On Node Expand.....	25-58
Suppress Grid Line	25-59
Suppress Node Indent/Outdent	25-60
Suppress Node Move Up/Down	25-61
Update Grid Buffer Row	25-62
Was Grid Cell Value Entered	25-63

26 Understanding Push Button Controls

26.1 Push Button Controls	26-1
26.2 Push Button Events.....	26-1

27 Understanding Radio Button Controls

27.1 Radio Button Controls.....	27-1
27.2 Radio Button Design-Time Considerations	27-1
27.3 Radio Button Events	27-1

28 Understanding Static Text Controls

28.1 Static Text Controls.....	28-1
--------------------------------	------

29 Using Subforms and Subform Aliases

29.1 Understanding Subforms.....	29-1
29.2 Understanding Subform Design-Time Considerations.....	29-2
29.3 Understanding Subform Events	29-3
29.4 Understanding Subform Runtime Processing.....	29-4
29.4.1 Control Initialization	29-4
29.4.2 Subform Push Buttons	29-4
29.4.2.1 Find.....	29-5
29.4.2.2 Select.....	29-5
29.4.2.3 Clear.....	29-5
29.4.2.4 Delete.....	29-5
29.4.2.5 Save.....	29-5
29.5 Creating Subforms	29-6
29.5.1 Understanding Subform Creation.....	29-6
29.5.2 Creating a Subform without a Power Form.....	29-6
29.5.3 Creating a Subform on a Power Form	29-6
29.5.4 Creating a Subform as a Tab Page.....	29-7
29.6 Reusing Subforms	29-7

29.6.1	Understanding Subform Reuse	29-7
29.6.2	Reusing a Subform on a Power Form	29-8
29.7	Working with Data Structures and Subforms	29-8
29.7.1	Mapping a Parent's Variables to a Child Subform	29-8
29.8	Working with Functions and Subforms	29-9
29.8.1	Adding a Function to a Subform	29-9
29.9	Subform System Functions	29-9
	Call Function.....	29-10
	Enable Subform	29-11
	Disable Subform	29-12
	Hide Subform	29-13
	Show Subform	29-14
	Update Parent.....	29-15
	Notify Parent	29-16
	Get Error Count.....	29-17
	Get Warning Count.....	29-18
	Get Subform ID	29-19
	Notify Child	29-20
	Trigger Default Action	29-21
	Expand Subform	29-22
	Collapse Subform.....	29-23

30 Understanding Tab and Tab Page Controls

30.1	Understanding Tab and Tab Page Controls.....	30-1
30.2	Creating Tab Controls	30-1
30.2.1	Creating a Tab Control.....	30-1
30.3	Tab Control System Functions	30-2
	Disable Tab Page.....	30-3
	Enable Tab Page	30-4
	Hide Tab Page	30-5
	Set Current Tab Page	30-6
	Set Tab Page Text	30-7

31 Understanding Text Block Controls

31.1	Text Block Controls.....	31-1
31.2	Charts in Text Blocks	31-1
31.3	Text Block Control Design-Time Considerations.....	31-2
31.4	Text Block Control Charts Design-Time Considerations.....	31-2
31.5	Text Block Events	31-3
31.6	Text Block Control System Functions	31-3
	Add Segment	31-4
	Get Last Clicked Segment.....	31-5
	Get Segment Information.....	31-6

Remove Segment.....	31-7
Update Segment.....	31-8
32 Understanding Secured Enterprise Search	
32.1 Secured Enterprise Search	32-1
33 Understanding Tree Controls	
33.1 Tree Controls	33-1
33.2 Tree Control Events	33-2
33.3 Tree Control System Functions...	33-2
Bulk Tree Load	33-3
Contract Tree Node	33-4
Delete Tree Node	33-5
Expand Tree Node	33-6
Get Node Information.....	33-7
Get Node Level.....	33-8
Get Tree Node Handle	33-9
Insert Tree Node.....	33-10
Set Bitmap Scheme.....	33-11
Set Node Bitmap	33-12
Set Node Information.....	33-13
Set Node Text.....	33-14
Set Tree Node Handle	33-15
34 Understanding Wizard Controls	
34.1 Wizard Controls	34-1
34.2 Wizard Control Design-Time Considerations	34-3
34.2.1 Implementing Re-entry Save.....	34-4
34.3 Wizard Control Events.....	34-4
34.4 Wizard Control Runtime Processing	34-6
34.4.1 Initialization.....	34-6
34.4.2 Page Entry	34-7
34.4.3 Next Button Processing.....	34-8
34.4.4 Previous Button Processing.....	34-9
34.4.5 Jumping Up- and Downstream	34-11
34.4.6 Save for Re-entry Button Processing.....	34-12
34.4.7 Cancel Button Processing	34-13
34.4.8 Finish Button Processing	34-13
34.5 Wizard Control Transaction Processing.....	34-16
34.6 Wizard Control System Functions.....	34-17
Get Current Wizard Page ID	34-18
Get Wizard Page Index	34-19
Set Selected Wizard Page.....	34-20

Set Wizard Form Mode	34-21
Set Wizard Page Index	34-22
Set Wizard Page Status.....	34-23
Suppress Wizard Page Validation and Save.....	34-24

A System Functions in Form Design Aid

A.1 System Functions	A-1
Control.....	A-2
Clear Control Error	A-3
Disable Control.....	A-4
Enable Control.....	A-5
Go to Url.....	A-6
Hide and Reclaim Space	A-7
Hide Control	A-8
Set Control Error	A-9
Set Control Text.....	A-10
Set Data Dictionary Item.....	A-11
Set Data Dictionary Overrides	A-12
Set Statusbar Text.....	A-13
Show Control	A-14
Was Value Entered	A-15
Expand Group Box	A-16
Collapse Group Box.....	A-17
General.....	A-18
Cancel User Transaction	A-19
Continue Custom Data Fetch.....	A-20
Copy Currency Information.....	A-21
Dynamic Form Interconnect (Web Only)	A-22
Get VCard (Release 9.2.1).....	A-23
Launch Batch Application	A-24
Launch Processing Options Dialog	A-26
Press Button	A-27
Run Executable.....	A-28
Send Email (Release 9.2.1)	A-29
Send Meeting Request (Release 9.2.1).....	A-30
Set Control Focus	A-31
Set Form Title.....	A-32
Dynamic Form Interconnect (Web Only)	A-33
Set Modified Web Object Behavior	A-34
Set Time Zone On Form	A-46
Set VCard (Release 9.2.1)	A-47
Stop Processing.....	A-48

Suppress Add	A-49
Suppress Default Visual Assist Form.....	A-50
Suppress Delete.....	A-51
Suppress Find	A-52
Suppress Update	A-53
Time Between	A-54
Was Form Record Fetched.....	A-55
Messaging	A-56
Send Message Extended.....	A-57
Mail Merge & Doc Gen (Web Only).....	A-60
Delete Document.....	A-61
Display Document	A-62
Download Template	A-63
Download Template for Doc Gen.....	A-64
Get XML Data Model	A-65
Run Doc Gen and Display	A-66
Run Mail Merge and Display	A-67
Run Multiple Mail Merge	A-68
Upload Template	A-69
Upload Template for Doc Gen.....	A-70

Glossary

Preface

Welcome to the *JD Edwards EnterpriseOne Tools Form Design Aid Guide*.

This guide has been updated for JD Edwards EnterpriseOne Tools releases 9.2.0.5, 9.2.1, and 9.2.1.4.

Audience

This guide is intended for developer and technical consultants who are responsible for assembling, building, and deploying applications.

This guide assumes you have a working knowledge of the following:

- The principles and customary practices of your business area.
- Working knowledge of Object Management Workbench.

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>

This guide contains references to server configuration settings that JD Edwards EnterpriseOne stores in configuration files. See the *JD Edwards EnterpriseOne Tools Server Manager Guide* for a list of configuration group settings for a server.

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..
<i>Italics</i>	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..

Introduction to JD Edwards EnterpriseOne Tools Form Design Aid

This chapter contains the following topics:

- [Section 1.1, "Development Tools Form Design Aid Overview"](#)
- [Section 1.2, "Development Tools Form Design Aid Implementation"](#)

1.1 Development Tools Form Design Aid Overview

JD Edwards EnterpriseOne Tools Form Design Aid is used to create or modify JD Edwards EnterpriseOne interactive applications. Interactive applications are composed of forms, and a form is the interface between a user and a table. This interface should present the data logically and contain the functions that are necessary to enter and manipulate data.

You can open multiple FDA projects. You are able to use all of FDAs functionality on each project that you have open. The number of projects that you can have open depends on your computer's performance capabilities. You cannot open the same application in two different FDA windows.

If you want to open more than one application in FDA, follow this process:

1. From OMW, choose an item to open in FDA and click the Design button.
2. On Object Management Workbench – [Interactive Application Design], click the Design Tools tab.
3. Click Start Form Design Aid.
4. On the toolbar at the bottom of your screen, click the minimized Object Management Workbench – [Interactive Application Design].
5. Click the Cancel button.
6. From OMW, choose an item to open in FDA and click the Design button.
7. On Object Management Workbench – [Interactive Application Design], click the Design Tools tab.
8. Click Start Form Design Aid.

1.2 Development Tools Form Design Aid Implementation

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Tools Form Design Aid.

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

1.2.1 JD Edwards EnterpriseOne Tools Form Design Aid Implementation Steps

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Tools Form Design Aid.

In the planning phase of your implementation, take advantage of all JD Edwards sources of information, including the installation guides and troubleshooting information. A complete list of these resources appears in the preface in *About This Documentation* with information about where to find the most current version of each.

This table lists the steps for JD Edwards EnterpriseOne Form Design Aid implementation.

- Set up default project in Oracle's JD Edwards EnterpriseOne Tools Object Management Workbench (OMW).
See "Configuring JD Edwards EnterpriseOne OMW," "Understanding JD Edwards EnterpriseOne OMW Configuration" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide*.
- Configure OMW transfer activity rules and allowed actions.
See "Configuring User Roles and Allowed Actions" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide*.

2

Working with Forms

This chapter contains the following topics:

- [Section 2.1, "Understanding Forms"](#)
- [Section 2.2, "Understanding Form Interconnections"](#)
- [Section 2.3, "Configuring Forms at Design Time"](#)
- [Section 2.4, "Creating Forms"](#)
- [Section 2.5, "Working with the Accessibility Violation Check Feature"](#)
- [Section 2.6, "Working with Quick Form"](#)

2.1 Understanding Forms

Use Oracle's JD Edwards EnterpriseOne Tools Form Design Aid (FDA) to create one or more forms for an application. A form is a graphical user interface where users interact with the system. A form can be used to search and display data, as well as enter new data and modify existing data.

A single application can contain one or more forms. Usually, a find/browse form is the first form that appears in the application. It enables the user to locate a specific record with which to work. Upon selecting a record, a subsequent form such as a fix/inspect form can be used to provide details of the record. With the introduction of power forms, applications can use one single power form to locate a specific record and display its detail records on one form.

A form has these elements:

- Form type

The form type establishes the basic purpose of a form. Each form type has default controls and processes.

- Business view

In an application, a business view (BV) links forms and tables efficiently by providing access only to that data required by the application. For example, if you have two tables with twenty columns each and the application only needs to access one column from one table and two columns from the other, you can make a BV that contains only those three columns. The application is more efficient because searches are limited only to those three columns, but the application still updates the actual tables when necessary. You must associate all forms, except the message form, with a BV.

- Controls

All objects on a form are controls. Controls include grids, check boxes, radio buttons, push buttons, subforms, and more.

- Properties

Properties define the appearance and function of the application, the forms in the application, and each control on each form.

- Data structure

A data structure defines the data that can be passed between forms. Data in the form data structure can be passed in or out of the form.

- Event rules

Event rules (ER) can contain processing instructions for specific events. Events are actions that occur on a form, such as clicking a button or using the Tab key to move out of a field. Use ER to attach business logic to any event. Events are triggered either as a result of user interaction with a control, such as clicking a button, or as a result of a system-controlled process, such as loading a grid.

See *JD Edwards EnterpriseOne Tools Event Rules Guide*.

JD Edwards ER Compare tool provides a side-by-side, visual comparison of an application's event rules, and another version of the event rules in another location. For example, ER Compare lets you compare a modified application to the original version of that application on the server or in an ESU.

See "JD Edwards EnterpriseOne ER Compare" in the *JD Edwards EnterpriseOne Tools Software Updates Guide*.

2.2 Understanding Form Interconnections

You can call a form from a form. This kind of form interconnection falls into three categories:

- Modal interconnections enable the user to view only one form at a time. After the child form begins, the user cannot access the parent form until the child form is closed.

Additionally, the data connection between the parent and child is usually static. Input data structure items are populated when the child form is launched, and output data structure items are populated when the child form closes. Parent form ER that follows a modal form interconnection call executes after the child form is closed.

In addition to static data connections, you can create dynamic form interconnections. A dynamic form interconnection enables you to call a form by passing the application and form ID. A dynamic form interconnection is always modal.

- Modeless interconnections enable the user to view multiple forms at the same time. After the child form is started, users can switch back and forth between the parent form and the child form. Additionally, data changes on a parent or child form are immediately reflected in all other open forms in the connection.

- Pop-up interconnections are similar to modal interconnections in that they enable the user to only work in one form at a time. After the child form is launched, the user cannot access the parent form until the child form is closed. However, the child form appears in a pop-up window, so that the parent form is still visible behind it, providing the user some context. (Release 9.2.1)

Modal is the default interconnection type. Modal interconnections are appropriate when you want to lead the user through a particular process in which a number of values must be input in a specific order. In this case, you want the user to completely fill out each form before moving on to the next one. Add and copy functions also lend themselves to modal processing because you want the user to complete the function before going on to others.

Modeless interconnections are valuable when the user needs to view or update a series of data records. Avoid using modeless form interconnections if both the parent and child forms should be presented to the user at the same time. A power form is more appropriate in this case.

The parent form in a modeless interconnect must be a find/browse form. The child form type can be fix/inspect or transaction forms (header and headerless detail). When the user updates a record on the transaction form, the parent find/browse form automatically reflects the change.

When the user closes the parent form, the system closes all its modeless children forms.

Parent form ER that follow a modeless form interconnection call execute immediately instead of waiting for the child form to return.

Release 9.2.1

Pop-up interconnections are useful when you want a modal interconnection, but do not want to lose sight of the parent form. By using a pop-up form and keeping the parent form visible, the user may better understand the context in which he is using the pop-up form.

Just like modal interconnections, pop-up interconnections are available for all form types. However, if you use a pop-up interconnection when calling an external form, it will not be honored at runtime. The external form will always display as a full screen.

2.3 Configuring Forms at Design Time

The property settings for a form control its appearance, how it displays errors, and how it interacts with its underlying business views. When you first create a form, the system prompts you to configure its properties. You can change the properties of a form later in the design process.

Some property values are common to all form types, although many are shared by just a few. This table lists the property values for all form types. The descriptions given in some cases are for general use only. If a particular property significantly impacts a given form type, then those impacts are discussed in detail in the section devoted to that form type.

Property	Form Type	Description
Business View Name	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Parent/Child Browse Power Edit Power Browse Edit Subform Browse Subform Edit Portlet Browse Portlet	The BV that is attached to the form.
Data Structure	All form types	The data structure underlying the form that maps incoming and outgoing data.
Enable In-Your-Face-Error Display	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	An option which, when selected, displays error text in red at the top of forms displayed in Web applications. Typically, the system indicates an application error by highlighting the Errors and Warnings link in the upper right area of the application. Selecting In-Your-Face-Errors has no noticeable effect on performance.
End Form on Add	Fix/Inspect Header Detail Headerless Detail	An option which, when selected, causes the system to close the form and return to the previous form after a user adds a record and clicks the OK button. If you want greater control over form flow, clear this option.
Entry Point	Find/Browse Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse With Release 9.2.1, External Form	An option which, when selected, flags the form as being the one that you want users to see when they first launch the application. If you do not assign one of the forms as the entry point, the runtime engine loads the first form it finds in the application. FDA does not permit you to set more than one form as an entry point. Fix/Inspect forms cannot be used as entry points.

Property	Form Type	Description
Fetch on Form Business View	Find/Browse Fix/Inspect Header Detail Search & Select Parent/Child Browse Edit Subform Browse Subform Edit Portlet Browse Portlet	An option which, when selected, causes the system to perform fetches based on information in the business view underlying the form. This option is unavailable if the form has no business view. Even though this setting is available for all form types, it only applies to Fix/Inspect, header/detail, headerless/detail, Power Edit forms with no grids, and subforms.
Fetch on Grid Business View	Find/Browse Headerless Detail Header Detail Search & Select Parent/Child Power Edit Power Browse	An option which, when selected, causes the system to update the grid during runtime based on information in the tables underlying the grid when the user clicks the OK button. This option is unavailable for entry unless the form contains a grid control.
Form Guide	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	The height and width in pixels to set the form guides. FDA indicates the optimum form size for different platforms by superimposing light blue lines that indicate height and width along the top and left-hand side of the form.
Form Name	All form types	The system name for the form. FDA names the form based on JD Edwards EnterpriseOne naming standards (forms start with W, subforms with S), the application name, and the creation sequence (the first form is appended with an A, the second with a B, and so forth). You cannot change this property.

Property	Form Type	Description
Form Type	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	The type (find/browse, search & select, and so forth) of the current form. You cannot change this property.
Height Width	All form types	The height and width of the form in dialog units. You can change the form size by resizing it manually with the mouse or by using these properties to set the size precisely.
Hover Form	Power Browse	This property, when selected, identifies the Power Browse form as a Hover Form. When selected, other options in Settings are no longer available. (Release 9.2.1)
Mapping Links	Power Edit Power Browse Edit Subform Browse Subform Edit Portlet Browse Portlet	The mapping of data between a parent and its child subforms.
Tile Wallpaper	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	This option, when selected, displays multiple copies of the background image in a tile-like manner. If you select this option, then you cannot select the Wallpaper property. Additionally, you cannot select this property unless you have set either the Wallpaper File or the Wallpaper Full Name File property.
Title	All form types	The text name of the form. This name appears at the top of the form when users work with the application. By default, FDA enters the form type in this field. You should change it.

Property	Form Type	Description
Total Controls on a Form	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	A field that shows the current number of controls (including subforms) on the selected form. You cannot change this property
Transaction	Fix/Inspect Header Detail Headerless Detail Power Edit Power Browse Edit Subform Edit Portlet	An option which, when selected, causes runtime to commit all changes at one time instead of individually. If the form makes a single database change or a group of unrelated changes, do not enable Transaction. However, if the form has a group of inserts that rely on each other and if the system should revert to the previously committed changes if a change fails or for another condition, then select Transaction.
		Transaction works differently for subforms and portlet forms.
Update on Form Business View	Fix/Inspect Header Detail Power Edit With No Grid Edit Subform Edit Portlet	An option which, when selected, causes the system to update the tables underlying the form (except those underlying the grid control) during runtime when the user clicks the OK button. This option is unavailable for entry unless you have attached a BV to the form.
Update on Grid Business View	Header Detail Headerless Detail Power Edit with Grid	An option which, when selected, causes the system to update the tables underlying the grid during runtime when the user clicks the OK button. This option is unavailable for entry unless the form contains a grid control.

Property	Form Type	Description
Wallpaper	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	An option which, when selected, displays a single image on the form as its wallpaper. If you enable this option, then you cannot enable the Tile Wallpaper property. Additionally, you cannot select this property unless you have set either the Wallpaper File or the Wallpaper Full File Name property.
Wallpaper File	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	The name of the image to use in conjunction with the Tile Wallpaper and Wallpaper properties. Use this property if the image is in the path code for the application.
Wallpaper Full File Name	Find/Browse Fix/Inspect Header Detail Headerless Detail Search & Select Message Form Parent/Child Browse Power Edit Power Browse	The name of the image to use in conjunction with the Tile Wallpaper and Wallpaper properties. Use this property if the image is not in the path code for the application.

2.4 Creating Forms

This section provides an overview of form creation and discusses how to:

- Create a form.
- Create a modal form interconnection.
- Create a dynamic modal form interconnection.
- Create a modeless form interconnection.
- Create a pop-up form interconnection (Release 9.2.1).

2.4.1 Understanding Form Creation

Use FDA to create one or more forms that appear in an application. These forms are the visual interface for the end user of the application and enable that user to view, add, or modify data that is stored in one or more tables.

After you create the form, you can modify the system data for the form, such as its metadata, Help ID, and so forth.

2.4.1.1 Recommended FDA Configuration

FDA offers a variety of tool bars and panes that you can place on the desktop while you work. This list provides recommendations for using these objects:

- Display the Main Toolbar, Layout, and Insert Controls tool bars at all times.
 - Use the Insert Controls tool bar to insert controls on a form.
 - Display the Application Tree View whenever possible.
- The pane helps you keep the entire application in mind, and it provides an easy way to open other objects in the application.
- Display the Property Browser at all times and use it to set the properties for all objects.
 - Display the Tab Sequence Toolbar when working with power forms and subforms.

This table lists the objects, why you would use them, and how to display them:

Object	Notes	Navigation in FDA
Main Toolbar	Provides standard actions such as open, cut, and paste.	View, Toolbars, Main Toolbar
Layout tool bar	Provides functions to align and space form controls precisely.	View, Toolbars, Layout
Insert Controls tool bar	enables you to place controls on a form with the push of a button.	View, Toolbars, Insert Controls
Status Bar	Displays a status bar at the bottom of the tool.	View, Status Bar
Application Tree View	Displays a pane showing the objects in the current application and their relationship to each other. Double-click an object in this pane to open it. The pane can be filtered for language. You can also configure the tree to display its hierarchy by control or by business unit.	View, Application Tree View
Property Browser	Displays a pane showing the properties of the selected object. You can change object properties in this pane.	View, Property Browser
Data Dictionary Browser	Displays a pane where you can search for data dictionary (DD) objects. You can drag and drop DD objects from this pane to the form.	View, Data Dictionary Browser
Business View Columns Browser	Displays a pane where you can search for columns in a BV. You can drag and drop objects from this pane to the form.	View, Business View Columns Browser

Object	Notes	Navigation in FDA
Tab Sequence Toolbar	enables you to manipulate the tab sequence functionality, including changing the object being sequenced on the form.	View, Toolbars, Tab Sequence Toolbar

2.4.2 Creating a Form

This task provides a general overview of the typical form creation process. Many of the steps in this task are described in greater detail in other topics.

To create a form:

1. In Oracle's JD Edwards EnterpriseOne Object Management Workbench (OMW), create an application or open an existing one, and then start FDA.
2. On Form Design Aid, select the type of form you want to create.
3. Configure the form properties as appropriate.
4. If required, attach one or more BVs to the form by selecting Form, Business View, Add Business View.

Headerless detail forms permit you to attach two BVs to them; all other forms, except for message forms, permit you to attach only one. You cannot attach a BV to a message form.

5. Add and configure controls, as required.

Set the Data Item Information property for a control to attach DD items or BV columns to it. To attach ER to a control, right-click the control and select Event Rules. To set text variables for a control, right-click the control and select Text Variables.

6. Arrange the controls on the form so that they line up and are equally spaced, and then resize the form to fit.
7. Add menu and tool bar exits to the form.
8. Save and test the form.

2.4.3 Creating a Modal Form Interconnection

To create a modal form interconnection:

1. On Event Rules Design, select an event.
2. Click the Form Interconnect button.
3. On Work with Applications, select the application to which you are connecting. Work with Forms displays available forms for the chosen application.
4. Select the form to which you want to connect (the target).
5. Select the 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 BV are automatically set up as the data structure.

6. Select Default (modal) from the Form Interconnect drop-down list. (Release 9.2.1)
7. In the Available Objects column, select the object that you want to pass and move it to the Data Structure-Value Column.

8. Indicate the direction of data flow between Value and Data Items.
As you click the direction arrow, it toggles through these 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.
9. Select the Include in Transaction option to include this interconnection for transaction processing.

This option is available for entry only if you are calling from a fix/inspect, header detail, or headerless detail form.

10. Click one of these buttons to add notes:

- Structure Notes
- Parameter Notes

11. After the data structure is defined, click the OK button.

Event Rules Design displays the form interconnect with this statement:

```
Call (Application <name> Form <name>)
```

2.4.4 Creating a Dynamic Modal Form Interconnection

To create a dynamic form interconnection:

1. On Event Rules Design, select an event.
2. Click the System Function button.
3. Select the General folder.
4. Select Dynamic Form Interconnect (Web Only).
5. Assign event rules to the input values **Application ID**, **Form ID** (both are required), **Version**, and **DSValue** (both are optional).

If you are not using either **Version** or **DSValue**, you must include the pipes around the null values.

6. Click OK to save the interconnection.

2.4.5 Creating a Modeless Form Interconnection

To create a modeless form interconnection:

1. On Event Rules Design for the find/browse form that you want to use (the source), select an event.
2. Click the Form Interconnect button.
3. On Work with Applications, select the application to which you are connecting.
Work with Forms displays available forms for the chosen application.
4. Select the 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. Select the version 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 BV, are automatically set up as the data structure.

6. Select the Modeless option. With Release 9.2.1, select Modeless from the Form Interconnect drop-down list.
7. In the Available Objects column, select objects that you want to pass and move them to the Data Structure-Value column.
8. Indicate the direction of data flow between Value and Data Items.

As you click the direction arrow, it toggles through these 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 these buttons to add notes:
 - Structure Notes
 - Parameter Notes

10. After you define the data structure, click the OK button.

Event Rules Design displays the form interconnect with this statement:

```
Call http://ple-mbarerra:8080/PSOL/eltools896pbr0/eng/psbooks/index.htm  
(Application <name> Form <name>)
```

2.4.6 Creating a Pop-up Form Interconnection (Release 9.2.1)

To create a pop-up form interconnection:

1. On Event Rules Design, select an event.
2. Click the Form Interconnect button.
3. On Work with Applications, select the application to which you are connecting.
Work with Forms displays available forms for the chosen application.
4. Select the form to which you want to connect (the target).
5. Select the 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 BV are automatically set up as the data structure.

6. Select Pop-up from the Form Interconnect drop-down list.
7. In the Available Objects column, select the object that you want to pass and move it to the Data Structure-Value Column.
8. Indicate the direction of data flow between Value and Data Items.

As you click the direction arrow, it toggles through these 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.
- 9.** Select the Include in Transaction option to include this interconnection for transaction processing.

This option is available for entry only if you are calling from a fix/inspect, header detail, or headerless detail form.

- 10.** Click one of these buttons to add notes:

- Structure Notes
- Parameter Notes

- 11.** After the data structure is defined, click the OK button.

Event Rules Design displays the form interconnect with this statement:

Call (Application <name> Form <name>)

Note: If you use a pop-up interconnection when calling an external form, it will not be honored at runtime. The external form will always display as a full screen.

2.5 Working with the Accessibility Violation Check Feature

This chapter provides an overview of the accessibility violation check feature and discusses how to:

- Enable the Accessibility Violation Check feature.
- Identify empty tooltips using the Accessibility Violation Check feature.

2.5.1 Understanding Accessibility Violation Check Feature

Accessibility Violation Check enables the EnterpriseOne application to capture the Accessibility violations at design time. It provides a way in the Form Design Aid (FDA) to implement empty tooltip validation on image controls. Accessibility Violation Check is developed in FDA wherein the EnterpriseOne application generates a warning report for all the image controls that have an empty tooltip.

The Accessibility Violation Check feature is available in 8.98 tools release only with JD Edwards EnterpriseOne 9.0 Apps release and above.

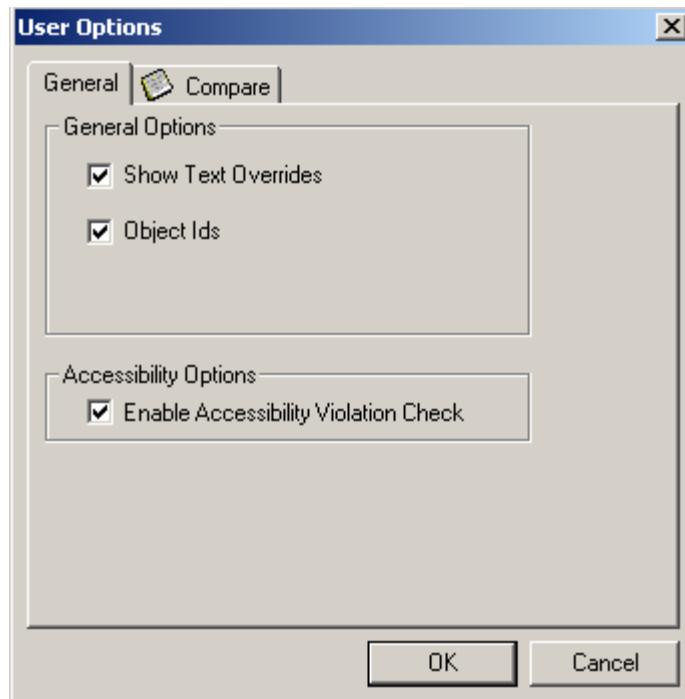
2.5.2 Enabling the Accessibility Violation Check Feature

You must enable the Accessibility Violation Check in EnterpriseOne in order to access it. To enable the Accessibility Violation Check feature:

1. In FDA, click the View menu, and then select User Options.

2. On User Options, click the General tab.
3. Select the Enable Accessibility Violation Check option, as shown in this example:

Figure 2–1 User Options, Enable Accessibility Violation Check



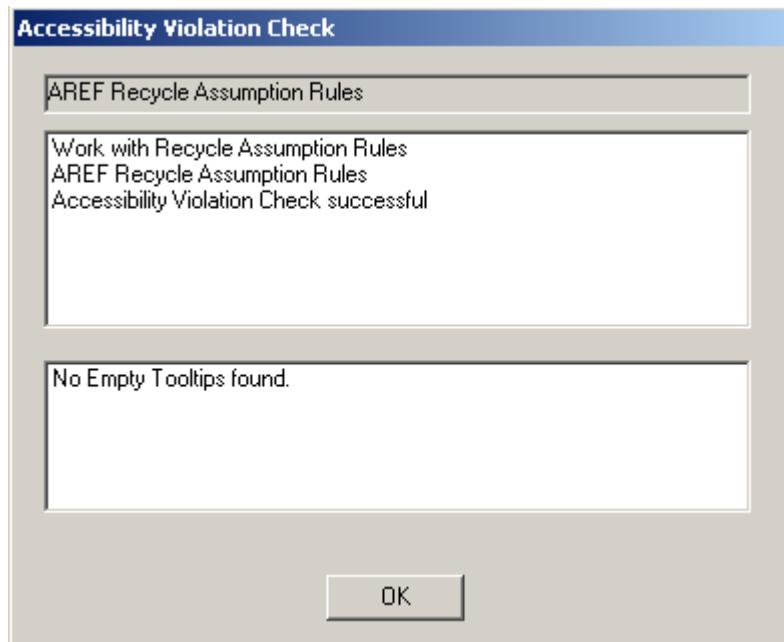
4. Click OK.

The check box for the Enable Accessibility Violation Check is selected by default. If you want to disable the feature, you must clear the Enable Accessibility Violation Check option.

2.5.3 Identifying Empty Tooltips Using Accessibility Violation Check Feature

To ensure that all the applications developed in EnterpriseOne have tooltips inserted on image controls, perform the following steps:

1. Verify that the Accessibility Violation Check option is selected on the User Options screen.
2. Click File, and then select Check Accessibility Violation Check.
3. EnterpriseOne performs a violation check for all the image controls that have an empty tooltip and generates the Accessibility Violation Check screen.
4. If all the image controls have tooltips inserted, EnterpriseOne displays the No Empty Tooltips found message, as shown in this example:

Figure 2–2 Accessibility Violation Check, No Empty Tooltips

Note: If the image controls have empty tooltips, EnterpriseOne displays details of the warnings and the location of the log file.

5. Click Yes on the Accessibility Violation Check screen to view the log files for the warnings generated.

The log file generated provides information about:

- Application Name
- Application Title
- Form Name
- Warning Description
- Bitmap Control ID
- Bitmap ER Title

Note: If you close any given EnterpriseOne application with empty tooltips, then the system displays an error message. Click Yes to exit the application, click No to remain in the application.

2.6 Working with Quick Form

This section discusses how to use Quick Form. By default, a form is opened in update mode. This means the system will only update existing data records. If a form is launched by a form interconnect statement in the Add button, then the form will be opened in add mode. This means the system will add new data into a business view when the form is closed.

2.6.1 Prerequisite

Attach a BV to the form.

2.6.2 Using Quick Form

Quick Form enables you to place multiple database fields on a form faster than choosing each data item individually. Select one or more data items, and Quick Form automatically places the fields on the new form simultaneously.

Depending on the number of selected fields, you might need to resize the form or move and align fields to achieve the desired layout.

To use Quick Form:

1. On Form Design Aid, select Form, Quick Form.
2. On Quick Form layout, select the number of columns of controls you want on the form and whether you want the columns arranged horizontally or vertically.
3. Select the data items from the BV that you want to display and move them to the columns in the form pane.
4. Use the buttons under the columns in the form pane to order the data items.
5. Click the OK button to place the fields on the form.

Quick Form remains open so that you can adjust the arrangement by changing the columns per row and the vertical and horizontal placement.

3

Working with Form Controls

This chapter contains the following topics:

- [Section 3.1, "Understanding Form Controls"](#)
- [Section 3.2, "Understanding Form Control Design-Time Considerations"](#)
- [Section 3.3, "Understanding Chart Control"](#)
- [Section 3.4, "Attaching Data Items to a Control"](#)
- [Section 3.5, "Associating a Data Item Description with a Field"](#)
- [Section 3.6, "Grouping Controls"](#)
- [Section 3.7, "Setting the Tab Sequence of Controls on a Form"](#)
- [Section 3.8, "JD Edwards EnterpriseOne FDA Compare"](#)

3.1 Understanding Form Controls

Use form controls to provide specific functions within an application such as these:

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

Each form includes specific default controls, depending on the type of form that you are creating. However, you might need to add additional controls when you design the form. Choose from standard Windows graphical controls as well as JD Edwards EnterpriseOne custom controls.

All form controls are locked by default, this is seen by a "NO" cursor when mouse is hovered over the controls. The controls can be unlocked by clicking on the Form menu and selecting Lock Form Controls. This option will unlock the selected form for modifying controls. An information message box will display when you launch an application to indicate if the form is in lock mode.

When controls are hidden they are by default not viewable in FDA. To view the hidden controls click the Show Hidden Controls in the Form menu. Hidden controls will be displayed in blue italic text.

This table lists the controls, the forms on which they can be used, and their purpose:

Control	Valid Form Types	Description
Calendar	All Form Types	Use a calendar control to provide standard calendar capabilities to users which can be tied to some system events.
Check Box	All Form Types	Use one or more check boxes to provide the user with options that are not mutually exclusive.
Combo Box	All Form Types	Use combo box to provide user a drop down list of items.
Edit	All Form Types	Use edit fields to display data and to enable users to enter information for a specific instance of a data item.
Grid control	Browse Portlet Edit Portlet Find/Browse (default control) Header Detail (default control) Headerless Detail (default control) Power Browse (default control) Power Edit Reusable Browse Subform Reusable Edit Subform Search & Select (default control) Wizard (page only)	Use grids to display data and to enable users to enter information. Unlike an edit control, grid controls can show multiple data items and multiple table rows at once.
Group	All Form Types	Use this control to group other controls together visually.
Image	All Form Types	Use image controls to place a static or animated graphic on a form.
Media Object	All Form Types	Use media object controls to enable users to enter rich text and attach files to a form.
Parent Child	Browse Portlet Edit Portlet Reusable Browse Subform Reusable Edit Subform Parent/Child Browse (default control) Power Edit Wizard (page only)	Use parent child controls to present a hierarchical grid view or a tree view.
Push Button	All Form Types	Use a push button to initiate an action or a set of actions.

Control	Valid Form Types	Description
Radio Button	All Form Types	Use radio buttons to provide the user with sets of options. The radio buttons in each set are mutually exclusive.
Saved Query Control	Browse Portlet Edit Portlet Find/Browse Header Detail Headerless Detail Parent/Child Browse Power Edit Power Browse Reusable Browse Subform Reusable Edit Subform Search & Select Wizard (page only)	Use a saved query control to enable users to create and save data queries and to provide them with a set of queries from which to choose.
Static Text	All Form Types	Use static text as labels on the form.
Subform	Browse Portlet Edit Portlet Reusable Browse Subform Reusable Edit Subform Power Edit Power Browse Wizard (Wizard control only)	Use subforms to provide one BV and a group of controls associated with it. Place multiple subforms on a power form to provide multiple, collective data views on one form. The subform created this way will not be reusable by other forms or applications.
Subform Alias	Browse Portlet Edit Portlet Reusable Browse Subform Reusable Edit Subform Power Browse Power Edit Wizard	Use a subform alias to place a reusable subform on the form. A reusable subform is a subform with a data view and a set of controls associated with it.
Tab Control	All Form Types	Use tab controls to present a large number of controls on one or more tab pages. Power forms can have any number of tab controls, but all other forms are restricted to one. You cannot use tab control on a subform that is a tab page.
Tab Page	Only apply to tab controls	Use a tab page control to define one page in a tab control.
Text Block Control	All Form Types	Use text block controls to display free-form HTML text and plain text elements.

Control	Valid Form Types	Description
Text Search Control	Browse Portlet Edit Portlet Find/Browse Header Detail Headerless Detail Parent/Child Browse Power Edit Power Browse Reusable Browse Subform Reusable Edit Subform Search & Select Wizard (page only)	Use text search controls to enable full text searches against a generated index (as opposed to searching against the underlying BV).
Tree Control	All Form Types	Use tree controls to display a tree structure.
Wizard	Wizard (default control)	Use wizard controls to create self-directed applications. This specialized control is available only on wizard forms.

3.2 Understanding Form Control Design-Time Considerations

Some control property values are common to all controls, although many are shared by just a few. This table lists the property values for all control types. The descriptions given in some cases are for general use only. If a particular property significantly impacts a given form type, then those impacts are discussed in detail in the chapter devoted to that control. In addition to the standard controls you can add to a form using the Insert menu in FDA, this table includes grid columns which have their own property values separate from the grid itself and the standard menu properties such as OK, Save, Cancel, and so forth because they can act as push button controls. They are referred to as buttons in this table; for example, Select button.

Property	Control	Description
Allow Image Items	Media Object	An option to permit image items in the media object.
Allow OLE Items	Media Object	An option to permit OLE objects in the media object.
Allow RTF Text	Media Object	An option to permit RTF text to be included in the media object.
Allow Text Items	Media Object	An option to permit plain text objects in the media object. The plain text in the file is stored in the database.

Property	Control	Description
Allowed in Saved Query	Edit Grid column Saved Query Control	An option to enable users to include the control as a filter value for a saved query. The property is available for entry only when a BV item is associated with the control.
Alternate Grid Row Format	Grid String	An HTML string that provides values for formatting the grid differently from the default system grid formatting. The system uses this string only if the Use Alternate Grid Row Format property is selected. You can also choose the formatting at runtime with the Set Grid Row Format system function. Set it to <DEFAULT> to use the system grid formatting and <ALTERNATE> to use this HTML string instead. The system function can switch between formats regardless of whether Use Alternate Grid Row Format is enabled.
Always Hidden	Parent Child	An option to hide the grid portion of the control.
Automatic Scroll Horizontal	Edit	A property that indicates whether the user can see text that exceeds the width of the field. If you permit scrolling, you can choose to have the system automatically scroll ten characters to the right when the user types text, or you can display a horizontal scroll bar.

Property	Control	Description
Automatic Scroll Vertical	Edit	A property that indicates whether the user can see text that exceeds the height of the field when the Lines property is set to Multiple . If you permit scrolling, you can choose to have the system automatically scroll down a page when the user presses Enter, or you can display a vertical scroll bar.
Automatically Find on Entry	Grid Parent Child	An option to cause runtime to populate the grid automatically when the form is entered.
Business View Name	Grid Parent Child Subform Subform Alias	A property that indicates the BV underlying the control. In all cases except one, the BV for the control is the same as the one for the form. On a header detail form, the grid control may have a BV that is different than the one underlying the controls that comprise the header.
Button Type	Push Button	A property that indicates the button type: OK, Cancel, Yes, No, and so forth. For form types except Message, the only option is Other.
Calendar Day View Visible	Calendar	An option to enable users to access the view of the calendar that shows a single day at a time.
Calendar Month View Visible	Calendar	An option to enable users to access the view of the calendar that shows an entire month at once.
Calendar Week View Visible	Calendar	An option to enable users to access the view of the calendar that shows an entire week at once.

Property	Control	Description
Checked Value	Check Box Grid column	A property that is the value that the control returns when a user selects the control. The property applies to a grid column only when its Display Style property is set to Check Box .
Clickable	Grid column Icon grid column Image Static Text	An option to cause runtime to fire the Text Clicked event when the user clicks the control.
Client Edge	Group Box	An option to give the group box the appearance of depth.
Collapsible	Subform Subform Alias Group Box	An option to enable users to hide the content of the subform or group box, displaying only its header.
Column Header One	Grid column	A property that shows the text to be displayed in the first line of the column heading.
Column Header Two	Grid column	A property that shows the text to be displayed in the second line of the column heading.
Column Moved to Tree	Parent Child	A property that enables you to control which columns appear in the tree portion of the control.
Column Order	Grid Parent Child	A property that controls the order in which the grid columns appear in the grid, from left to right in English.
Column Sort Order	Grid Parent Child	A property that controls the order in which data returned to the grid is ordered for display.
Control ID	All control types	A property that shows the system ID of the current control. It cannot be changed.
Data Item Information	Check Box Combo Box Edit Grid column Radio Button Static Text	A property that shows the DD item or BV column associated with the control. BV column choices come from the BV associated with the form.

Property	Control	Description
Data Structure	Subform Subform Alias	A property that shows the data structure being used to pass data between parent and child. If no data structure is attached to the form, the property is unavailable for entry.
Default cursor on add mode	Edit Grid column Media Object	An option to designate this field as the one in which the cursor appears initially when the form appears in Add mode. You can select this option for only one edit control on any given form.
Default cursor on update mode	Edit Grid column Media Object	An option to designate this field as the one in which the cursor appears initially when the form appears in Update mode. You can select this option for only one edit control on any given form.
Disable Copy	Parent Child	An option to prevent users from using the copy function in a cut/copy/paste operation.
Disable Drag and Drop (Cut/Copy/Paste)	Parent Child	An option to prevent users from using the cut/copy/paste function.
Disable Move (Cut)	Parent Child	An option to prevent users from using the move/cut function in a cut/copy/paste operation.
Disable Page-at-a-Time Process	Grid Parent Child	An option to disable page-at-a-time processing. Page-at-a-time processing enables the runtime engine to fetch a single page of data only on the initial search call. If the user pages down, then runtime fetches only enough data to fill the next page. When disabled, runtime fetches and loads into memory all of the data at once. The number of rows that constitute a page of data is based on the Grid Row Count property.

Property	Control	Description
Disable QBE	Grid column	An option to disable the QBE cell above a given column.
Disabled	Calendar Check Box Close button Combo Box Delete button Edit Find button Grid Grid column Group Box Media Object OK button Parent Child Push Button Radio Button Saved Query Control Select button Static Text Subform Subform Alias Text Search Control	An option to disable the control, preventing user interaction with it (although it can still be seen). You can also enable and disable controls during runtime with the Enable Control and Disable Control system functions.
Display Customized Grid	Grid	An option to enable users the option to customize the grid through grid formats.
Display Export to Excel	Grid	An option to enable users the option to send the contents of the control to an Excel spreadsheet.
Display Export to Word	Grid	An option to enable users the option to send the contents of the control to a Microsoft Word file.
Display Import from Excel	Grid	An option to enable users the option to bring the contents of an Excel spreadsheet into the control.
Display Style	Grid column	An option to make cells in the column act (and appear) as check boxes.

Property	Control	Description
Do Not Clear After Add	Edit Grid column	An option to retain the data in the field after runtime performs an add function. After performing an add, runtime usually clears all form fields.
Editable	Subform Subform Alias	A property that indicates whether the subform type can be edited. Reusable edit subforms can be edited, reusable browse subforms cannot. Embedded subforms are editable or not based on their context within the parent form.
Expand All/Collapse All	Parent Child	An option to provide a button for the user to expand or collapse the entire tree.
Fetch on Form/Subform Businessview	Subform Subform Alias	An option to cause the system to perform fetches from the subform business view. This option is disabled unless you have attached a BV to the subform.
File Name	Image	A property that displays the name of the image file to be displayed in the control.
Filter Criteria	Edit Radio Button	A property that indicates whether the control value should be incorporated into the database fetch. If you want to use the value, then you must also choose the relational operator by which the value should be evaluated. In some cases, you can designate that the relational value should be chosen by the user at runtime instead.

Property	Control	Description
Filter Criteria - Checked Filter	Check Box	A property that indicates whether the control should be incorporated in the database fetch as a filter criteria when the check box is selected. If you want to use this control as a filter, then you must also define how the filter value will be used by providing a relational operator or designate that the relational value should be set by the user at runtime instead.
Filter Criteria - Unchecked Filter	Check Box	A property that indicates whether the control should be incorporated in the database fetch as a filter criteria when the check box is cleared. If you want to use this control as a filter, then you must also define how the filter value will be used by providing a relational operator or designate that the relational value should be set by the user at runtime instead.
Flat	Group Box	An option to give the group box border the appearance of height.
Form Name	Subform Subform Alias	A property that shows the system name for the subform. FDA names the subform based on JD Edwards EnterpriseOne naming standards (subforms start with S), the application name, and the creation sequence (the first subform is appended with an A, the second with a B, and so forth). You cannot change this property.
Full File Name	Image	A property that displays the name and location of the image file to be displayed in the control.
Grid Row Count	Grid	A property that shows the number of rows in a grid that constitute a "page."

Property	Control	Description
Height	Calendar Check Box Close button Combo Box Delete button Edit Find button Grid Group Box Image Media Object OK button Parent Child Push Button Radio Button Saved Query Control Select button Static Text Subform Tab Control Text Box Control Text Search Control Tree Control Wizard	A property that shows the height of the control in dialog units. You can change the height by typing a different value.
Hide HTML Row Selector	Grid	An option to hide the row selector when the form is viewed in HTML. You might choose to hide the selector either because it is not needed or because you want to prevent users from selecting entire rows.
Hide in Grid	Parent Child	A property that controls whether the designated tree column should be hidden in the grid. You should set this property to prevent the same column being displayed in two places, both in the tree and the grid.

Property	Control	Description
Hide Query By Example	Grid Parent Child	An option to hide the query-by-example (QBE) line above the grid. You might choose to hide the QBE either because it is not needed or because you want to prevent the users from defining query criteria on grid columns.
Indent and Outdent	Parent Child	An option to enable the user to change the horizontal position of a node in the tree.
Justification	Check Box Edit Radio Button Static Text	A property that indicates whether the text of the control will be left-, center-, or right-justified.
Key Relations	Parent Child	A property that defines the values to use for certain key fields when the system builds queries. When you map a child key to a parent key, then the system sets the value of the parent variable equal to the associated child value for purposes of filtering the query.

Property	Control	Description
Left	Calendar Check Box Close button Combo Box Delete button Edit Find button Grid Group Box Image Media Object OK button Parent Child Push Button Radio Button Saved Query Control Select button Static Text Subform Subform Alias Tab Control Text Box Control Text Search Control Tree Control	A property that shows the distance from the left edge of the form to the left edge of the control in dialog units. You can change the horizontal placement of the control by typing a different value.
Lines	Edit	An option to enable the field to display multiple lines of text. If you select this option, usually you also set the Automatic Scroll View property such that users can view the text if it exceeds the height of the control.
Load Text by Instance	Subform Alias	An option that defines how jargon will be loaded for reusable subforms. If selected, the system fetches jargon based on the form that is using the subform. If cleared, the system fetches jargon based on the application where the subform is defined.

Property	Control	Description
Location Indicator Feature	Parent Child	An option to enable or disable location indicator functionality for the parent child control. If selected, the system enables the user to choose to display a location indicator for each tree node.
Maintain Aspect Ratio	Image	An option to maintain the original width-to-height ratio when the image is resized.
Mapping Links	Subform Subform Alias	A property that shows the mapping of data between a parent and its child forms. You must define this mapping to pass data between parent and child forms.
Menubar Separator	Close button Delete button Find button OK button Select button	An option to display a line above the name of the control when viewed in a menu list.
Modal Frame	Group Box	An option to give the group box the appearance of height.
Move Up and Down	Parent Child	An option to enable the user to change the vertical position of a node in the tree.
Multi-Line Edit	Grid	An option to post rows in groups of three to five to the database in the background. When disabled, in low interactivity mode, runtime posts each time the user tabs out of the row, forcing the user to wait for a refresh before continuing. This option does not apply to high interactivity.
Multiple Select	Grid Parent Child	An option to enable users to select multiple lines to affect with a single operation, such as cut-and-paste.
New Text Item on Open	Media Object	An option to cause the control to create a new text item automatically when the user first opens the control.

Property	Control	Description
No Adds On Update Grid	Grid Parent Child	An option to prevent users from adding new records to the control. Users can still edit existing rows, however.
No display if currency is OFF	Check Box Edit Grid column Radio Button Static Text	An option to hide the control if currency is disabled.
Node ID Column	Parent Child	An option to use this column as a unique identifier for that row. The system functions, Insert Grid Buffer Row By Node ID and Get Related Node ID , require a node ID column to work correctly.
Number of columns joined to header	Grid Parent Child	A property that enables backwards compatibility of certain legacy applications. Note: Set this property value to zero for new applications.
Overrides Button	Check Box Combo Box Edit Grid column Radio Button	A button to set DD overrides for the control.
Parent	Subform Subform Alias	A property that indicates the form that acts as the parent to the current subform.
Password	Edit	An option to cause the system to display an asterisk in place of the character the user actually typed. This feature is most often used to help protect passwords.
Prevent Resizing	Image	An option to prevent the image from being resized during design time.
Position of Saved Query links	Saved Query Control	A property that indicates where the saved query links appear relative to the main body of the control itself.

Property	Control	Description
Process All Rows in Grid	Grid	An option to cause runtime to apply row changed and row exited logic to all rows, no matter their state.
Product Synch Mapping	Parent Child	A property that indicates how the data in the column will be used if the parent child control on which the grid resides has the Product Synch Mode property enabled.
Product Synch Mode	Parent Child	An option to use the control for process mapping applications (that is, rapid manufacturing).
Progress Indicator	Wizard	An option to display a progress indicator during runtime. You must also indicate its type, if the indicator is displayed.
Read Only	Edit Media Object	An option to prevent users from changing the value in the control.
Reclaim Whitespace	Grid	An option to cause runtime to shrink the control vertically so it displays only those rows which contain data (HTML only).
Re-entry Save	Wizard	An option to enable users to save their input, quit, and then re-launch the wizard later, starting at the point where they saved.
Required field	Combo Box Edit Grid column	An option to force users to enter a value into this control before being able to execute a form-level action such as OK or Save (except Cancel). Runtime displays an asterisk next to the required field label, if control and label are connected.

Property	Control	Description
Reusable	Subform Subform Alias	A property that indicates whether the subform is reusable. Subforms that were created independently of a parent (such as, reusable browse subform and reusable edit subform) are reusable and may be referenced by an alias on any number of power forms. Subforms that were created as a control on a parent form cannot be referenced with an alias and are therefore not reusable.
Show DD Alias Tooltip	Icon grid column	An option to display the Data Dictionary alias for the data item that an icon represents in the tooltip for that icon.
Show DD Name Tooltip	Icon grid column	An option to display the Data Dictionary name for the data item that an icon represents in the tooltip for that icon.
Show details of all tree nodes	Parent Child	An option to display one tree node for each grid row.
Show Header	Subform Subform Alias	An option to display the header of the subform during runtime.
Show Icon Tooltip	Icon grid column	An option to display the descriptive text for the data item that an icon represents in the tooltip for that icon.
Show Value Tooltip	Icon grid column	An option to display the value of the data item that an icon represents in the tooltip for that icon.
Sort Order	Grid column	A property that displays the order in which column data will be sorted (ascending or descending).
Sortable by End User	Grid column	An option to enable the user to reorder the grid contents based on this column.
Sorted	Combo Box	An option to sort the items in the drop-down box alphabetically.

Property	Control	Description
Static Edge	Group Box	An option to give the group box border the appearance of depth.
Subform Application Name	Subform Alias	A property that shows the name of the application that contains the reusable subform to which the alias points.
Support Aggregation	Grid column	An option to enable the aggregation of a math numeric or integer column type on a Find Browse form.
Support Multiple Currencies	Grid column	An option to enable the column to handle and display amounts in differing currency types.
Suppress Validation and Save	Wizard	An option to prevent runtime from validating and saving the data in the control.
Tab Stop	Calendar Check Box Combo Box Edit Grid Group Box Media Object Parent/Child Push Button Radio Button Saved Query Control Static Text Text Search Control	An option to enable users to press Tab to move the focus to the control.
Task List	Wizard	An option to display the task list during runtime.
Text is Overridden	Check Box Combo Box Edit Grid column Radio Button Static Text	If the control is associated with a BV column or DD item, an option to change the title (the text that users can see) in this instance.

Property	Control	Description
Title	Calendar Check Box Combo Box Edit Grid Group Box Image Media Object Parent Child Push Button Radio Button Saved Query Control Static Text Subform Tab Control Tab Page Text Block Control Text Search Control Tree Control Wizard	A property that shows the text that the user can see. If the control is associated with a particular BV column or DD item, then you might need to select an overrides option to enable the field if you want to change its displayed text in a given instance.
Tool Tip	Image	A property that shows the text that appears in the tool tip for the image. The tool tip is the text that appears when the user hovers over the object.
Toolbar	Close button Delete button Find button OK button Select button	An option to display the control as a button on the standard application tool bar.
Toolbar Separator	Close button Delete button Find button OK button Select button	An option to display a line to the left of the control button when it appears on the standard application tool bar.

Property	Control	Description
Top	Calendar Check Box Close button Combo Box Delete button Edit Find button Grid Group Box Image Media Object OK button Parent Child Push Button Radio Button Saved Query Control Select button Static Text Subform Subform Alias Tab Control Text Search Control Text Box Control Tree Control	A property that shows the distance from the top edge of the form to the top edge of the control in dialog units. You can change the vertical placement of the control by typing a different value.
Transaction	Subform	An option to cause runtime to commit all changes at one time instead of individually. If the subform makes a single database change or a group of unrelated changes, do not select Transaction. However, if the subform has a group of inserts that rely on each other and if the system should revert to the previously committed changes if a change fails or for another condition, then select Transaction.
Unchecked Value	Check Box Grid column	A property that is the value that the control returns when a user clears the control. The property applies to a grid column only when its Display Style property is set to Check Box .

Property	Control	Description
Update Mapping Link	Push Button	An option to cause the parent to push data to its child before runtime performs any button processing.
Update Mode	Grid	An option to make the grid input-capable. Runtime implements this option only when the grid resides on input-type forms such as fix/inspect or header detail.
Update on Form/Subform Businessview	Subform	An option to cause the system to update the tables underlying the subform (except those underlying the grid control) during runtime when the user clicks the OK button. If you want greater control over subform updates, clear this option. This option is unavailable for entry unless you have attached a BV to the subform.
Use Alternate Grid Row Format	Grid	An option to use the HTML string in the Alternate Grid Row Format property to format the grid instead of using the default system formatting for grids.

Property	Control	Description
UTC Display Format	Edit Grid control	A property that controls how to display time and date fields. This property value is available for entry if the BV column or DD item associated with the control has a type of U-Time. This type of data represents date and time in Coordinated Universal Time (UTC) format. The offset is from Greenwich Mean. This field is not a mask. Whatever format you select comprises the only data that is saved to the database. Therefore, if you select a format that displays only the date and not the time, then only the date is written to the database.
Value	Radio Button	A property that is the value that the control returns when a user selects the control.
Visible	Calendar Check Box Combo Box Edit Grid Grid column Group Box Image Media Object Parent Child Push Button Radio Button Saved Query Control Static Text Subform Subform Alias Tab Control Text Box Control Text Search Control Tree Control Wizard	An option to enable users to see the control. You can also hide and show the control during runtime with the system functions, Hide Control and Show Control .

Property	Control	Description
Width	Calendar Check Box Close button Combo Box Delete button Edit Find button Grid Grid column Group Box Image Media Object OK button Parent Child Push Button Radio Button Saved Query Control Select button Static Text Subform Tab Control Text Box Control Text Search Control Tree Control Wizard	A property that shows the width of the control in dialog units. You can change the width by typing a different value.
Wildcard	Edit	An option to enable users to use the asterisk character as a wildcard when performing searches.
Wrap Text	Grid column	An option to enable text to wrap if it exceeds the column width. Otherwise, the system truncates column text.

3.3 Understanding Chart Control

This section provides an overview of the Chart Control.

3.3.1 Using the Chart Control

Use the Chart Control to set properties on a visible chart. These charts help plant managers quickly quantify the state of operations at a plant. FDA provides a set of system functions specific for the Chart Control. The chart Control uses the following files:

- *Data XML*

The data XML file determines the information that a graph displays. The Chart Control uses a data XML file that you create to call system functions. You can create the data XML file using any editor, such as business functions or event rules. Data XML is required for the Chart Control. See Appendix: A for examples of data XML.

- *Graph XML*

The graph XML file determines how the information that a graph displays is formatted, i.e. in a bar or pie format. JD Edwards provides default templates of graph xml files for you to use to design your charts. You can change or modify the information in the graph XML to change the appearance of the chart. The graph XML file is optional for the Chart Control. If the graph XML file is not specified for the Chart Control, then the graph will be displayed in the format specified by the graph name attribute in the data XML file. See Appendix: B for examples of graph XML.

FDA enables you to create the following chart types:

- bar_basic
- combo_basic
- combo_markers
- ine_basic
- pie_basic
- pie_onetime
- stacked_bar_basic
- stacked_bar_ontime

3.3.2 Using XML Graph Access

To quickly view how your chart will look, you can use the Direct XML Graph Access application located at the following

URL: <http://<web server host>:<web server port>/jde/GraphPrototype.maf>

The Direct XML Graph Access contains a sample of every chart JD Edwards EnterpriseOne provides. It contains an example DTD and data XML for each chart that you can edit to see immediately how your changes will impact the chart. You can also enter your own DTD and data XML to review a chart you have created.

You must use your EnterpriseOne login credentials to access the Direct XML Graph Access application.

3.3.3 System Functions for Charting

The following table contains the system functions EnterpriseOne created for the charting control.

System Function	Description
Draw Chart(Chart Control)	Call this system function to render a chart. Assume its DataXML is correctly populated. After calling other system functions to set chart properties, be sure to call this system function to show the new result. Otherwise, the new properties will not be shown on the form.
Set Data XML(Chart Control, data)	Call this system function to set DataXML for the chart. Runtime will automatically read graphName from DataXML and set it to GraphName on this chart. The data XML is required to be set for a chart control. Otherwise an error is displayed.
Set Graph XML(Chart Control, graphXML)	Call this system function to set customized graphXML for the chart. If this function is called, its graphXML will be used instead of graphName from the DataXML.
Set Graph Template(Chart Control, graphTemplate)	Call this system function to set graphName for the chart. Runtime will read the checked in graph.xml based on the graphName.
Set Title(Chart Control, title)	Call this system function to set the title for this chart.
Set Title Visible(Chart Control, true/false)	Call this system function to make the title visible or invisible for this chart.
Set Footnote(Chart Control, footnote)	Call this system function to set a footnote for the chart.
Set Footnote Visible(Chart Control, true/false)	Call this system function to make a footnote visible or invisible for the chart.
Set Legend Orientation(Chart Control, Orientation)	Call this system function to set the orientation for this chart. Valid values for Orientation are <Vertical> or <Horizontal>.
Set Legend Position(Chart Control, position)	Call this system function to set the position for the chart. Valid values for Position are <Top>,<Left>,<Right>, or <Bottom>.
Set Legend Visible(Chart Control, true/false)	Call this system function to make the legend visible or invisible for the chart.
Reset(Chart Control)	Call this system function to reset all properties, but leave DataXML,graphXML and graphName unchanged.
Get Last Chart Click Event(rowIndex, columnIndex)	Call this system function at the beginning of the "Chart is Clicked" FDA event. Tools will return the actual row and column index of the element clicked in the output ER variables. Application development can then use these ER variable values to execute any subsequent logic in the "Chart is Clicked" FDA event.
Turn Off Chart Clicked Event(Chart Control)	Call this system function to disable clickable events on this chart. The default is to have clickable events be disabled.
Turn On Chart Clicked Event(Chart Control)	Call this system function to enable clickable events on this chart. The default is to have clickable events be disabled.

3.3.4 C Code System Functions

The following table contains the C code system functions that applications call within C business functions to set chart properties in the web client.

System Functions	Description
JDE_SetDataXML(lpBhvrCom, int chartId, JCHAR * dataXml)	Call this system function to set DataXML for this chart.
JDE_SetGraphXML(lpBhvrCom, int chartId, JCHAR * graphXml)	Call this system function to set customized graphXML for this chart.
JDE_DrawChart(lpBhvrCom, int chartId)	Call this system function to render a chart. Assume its DataXML is correctly populated to show the new result. Otherwise, the new properties won't be shown on the form.

C side system functions do not exist for the title, footnote, legend, and chart click properties.

3.3.5 Example Data XML

This section contains example data XML for each of the chart type provided in EnterpriseOne. You can modify the code in these data XMLs to alter what information is displayed on the chart.

The graph name attribute in the data XMLs below relates directly to the graph XML types in section Example Graph XMLs. All of the graph types share a common DTD, which is described in section Example Graph DTD.

bar_basic.xml

```
<Graph graphName="bar_basic">
<O1Title text="Week Ending" visible="true" />
<Y1Title text="Cost Variance (USD)" visible="true" />
<LocalRelationalData>
<Row columnKey="9/1/05" rowKey="Actual Variance" dataValue="1504" />
<Row columnKey="9/8/05" rowKey="Actual Variance" dataValue="980" />
<Row columnKey="9/15/05" rowKey="Actual Variance" dataValue="-675" />
<Row columnKey="9/22/05" rowKey="Actual Variance" dataValue="784" />
<Row columnKey="9/29/05" rowKey="Actual Variance" dataValue="0" />
<Row columnKey="9/1/06" rowKey="Actual Variance" dataValue="1504" />
<Row columnKey="9/8/06" rowKey="Actual Variance" dataValue="980" />
<Row columnKey="9/15/06" rowKey="Actual Variance" dataValue="-675" />
<Row columnKey="9/22/06" rowKey="Actual Variance" dataValue="784" />
<Row columnKey="9/29/06" rowKey="Actual Variance" dataValue="0" />
</LocalRelationalData>
</Graph>
```

combo_basic

```
<Graph graphName="combo_basic">
<O1Title text="Week Ending" visible="true" />
<Y1Title text="Days" visible="true" />
<LocalRelationalData>
<Row columnKey="1-Sept 05" rowKey="Actual DSI" dataValue="38" />
<Row columnKey="8-Sept 05" rowKey="Actual DSI" dataValue="25" />
<Row columnKey="15-Sept 05" rowKey="Actual DSI" dataValue="24" />
<Row columnKey="22-Sept 05" rowKey="Actual DSI" dataValue="27" />
<Row columnKey="1-Sept 05" rowKey="Target DSI" dataValue="10" />
<Row columnKey="8-Sept 05" rowKey="Target DSI" dataValue="10" />
</LocalRelationalData>
</Graph>
```

```

<Row columnKey="15-Sept 05" rowKey="Target DSI" dataValue="10" />
<Row columnKey="22-Sept 05" rowKey="Target DSI" dataValue="10" />      </Local⇒
RelationalData>
</Graph>
```

combo_markers.xml

```

<Graph graphName="combo_markers">
<O1Title text="Week Ending" visible="true" />
<Y1Title text="Days" visible="true" />
<LocalRelationalData>
<Row columnKey="1-Sept 05" rowKey="Actual DSI" dataValue="38" />
<Row columnKey="8-Sept 05" rowKey="Actual DSI" dataValue="25" />
<Row columnKey="15-Sept 05" rowKey="Actual DSI" dataValue="24" />
<Row columnKey="22-Sept 05" rowKey="Actual DSI" dataValue="27" />
    LocalRelationalData>
</Graph>
```

line_basic.xml

```

<Graph graphName="line_basic">
<O1Title text="Week Ending" visible="true" />
<Y1Title text="Cost Variance (USD)" visible="true" />
<LocalRelationalData>
<Row columnKey="9/1/05" rowKey="Actual Variance" dataValue="1504" />
<Row columnKey="9/8/05" rowKey="Actual Variance" dataValue="980" />
<Row columnKey="9/15/05" rowKey="Actual Variance" dataValue="-675" />
<Row columnKey="9/22/05" rowKey="Actual Variance" dataValue="784" />
<Row columnKey="9/29/05" rowKey="Actual Variance" dataValue="0" />
<Row columnKey="9/1/06" rowKey="Actual Variance" dataValue="1504" />
<Row columnKey="9/8/06" rowKey="Actual Variance" dataValue="980" />
<Row columnKey="9/15/06" rowKey="Actual Variance" dataValue="-675" />
<Row columnKey="9/22/06" rowKey="Actual Variance" dataValue="784" />
<Row columnKey="9/29/06" rowKey="Actual Variance" dataValue="0" />
</LocalRelationalData>
</Graph>
```

pie_basic.xml

```

<Graph graphName="pie_basic">
<LocalRelationalData>
<Row columnKey="January 2005" rowKey="On Time" dataValue=".11" />
<Row columnKey="January 2005" rowKey="Early" dataValue=".21" />
<Row columnKey="January 2005" rowKey="Late" dataValue=".05" />
<Row columnKey="January 2005" rowKey="Past" dataValue=".12" />
<Row columnKey="January 2005" rowKey="History" dataValue=".25" />
<Row columnKey="January 2005" rowKey="Now" dataValue=".03" />
<Row columnKey="January 2005" rowKey="Then" dataValue=".14" />
<Row columnKey="January 2005" rowKey="When" dataValue=".09" />
</LocalRelationalData>
</Graph>
```

pie_ontime.xml

```

<Graph graphName="pie_ontime">
<LocalRelationalData>
<Row columnKey="January 2005" rowKey="On Time" dataValue=".11" />
<Row columnKey="January 2005" rowKey="Early" dataValue=".21" />
<Row columnKey="January 2005" rowKey="Late" dataValue=".05" />
</LocalRelationalData>
</Graph>
```

stacked_bar_basic.xml

```
<Graph graphName="stacked_bar_basic">
<O1Title text="Week Ending" visible="true" />
<Y1Title text="On Time Production" visible="true" />
<LocalRelationalData>
<Row columnKey="9/1/05" rowKey="On Time" dataValue="95" />
<Row columnKey="9/8/05" rowKey="On Time" dataValue="8" />
<Row columnKey="9/15/05" rowKey="On Time" dataValue="9.2" />
<Row columnKey="9/22/05" rowKey="On Time" dataValue="70" />
<Row columnKey="9/1/05" rowKey="Early" dataValue="1" />
<Row columnKey="9/8/05" rowKey="Early" dataValue="10" />
<Row columnKey="9/15/05" rowKey="Early" dataValue="40" />
<Row columnKey="9/22/05" rowKey="Early" dataValue="18" />
<Row columnKey="9/1/05" rowKey="Late" dataValue="0" />
<Row columnKey="9/8/05" rowKey="Late" dataValue="15" />
<Row columnKey="9/15/05" rowKey="Late" dataValue="4" />
<Row columnKey="9/22/05" rowKey="Late" dataValue="50" />
</LocalRelationalData>
</Graph>
```

stacked_bar_ontime.xml

```
<Graph graphName="stacked_bar_ontime">
<O1Title text="Week Ending" visible="true" />
<Y1ReferenceLine>
<ReferenceLine index="0" visible="true" text="Goal" value="95.0" />
</Y1ReferenceLine>
<Y1Title text="On Time Production" visible="true" />
<LocalRelationalData>
<Row columnKey="9/1/05" rowKey="On Time" dataValue=".90" />
<Row columnKey="9/8/05" rowKey="On Time" dataValue=".85" />
<Row columnKey="9/15/05" rowKey="On Time" dataValue=".92" />
<Row columnKey="9/22/05" rowKey="On Time" dataValue=".72" />
<Row columnKey="9/1/05" rowKey="Early" dataValue=".10" />
<Row columnKey="9/8/05" rowKey="Early" dataValue=".10" />
<Row columnKey="9/15/05" rowKey="Early" dataValue=".04" />
<Row columnKey="9/22/05" rowKey="Early" dataValue=".18" />
<Row columnKey="9/1/05" rowKey="Late" dataValue="0" />
<Row columnKey="9/8/05" rowKey="Late" dataValue=".05" />
<Row columnKey="9/15/05" rowKey="Late" dataValue=".04" />
<Row columnKey="9/22/05" rowKey="Late" dataValue=".10" />
</LocalRelationalData>
</Graph>
```

3.3.6 Example Graph XMLs

This section contains example graph XML for each of the chart type provided in EnterpriseOne. You can modify the code in these graph XMLs to alter how the chart displays information.

bar_basic.xml

```
<?xml version="1.0" ?>
<Graph version="3.2.0.22" autoLayout="AL_ALWAYS" markerTooltipType="MTT_VALUES"⇒
graphicAntialiasing="true" textAntialiasing="true" seriesEffect="SE_AUTO_⇒
GRADIENT" frameSizeAutomatic="false">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
```

```

underline="false"⇒
/> </Footnote>
<LegendArea visible="true" automaticPlacement="AP_NEVER" position="LAP_TOP" fill⇒
Color="#ffffffff" />
<LegendText horizontalAlignment="LEFT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/> </LegendText>
<O1TickLabel textRotation="TR_HORIZ" horizontalAlignment="RIGHT" tickLabel⇒
Staggered="false" tickLabelSkipMode="TLS_AUTOMATIC" tickLabelSkipCount="0" tick⇒
LabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/> </O1TickLabel>
<O1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/> </O1Title>
<SeriesItems defaultColor="" />
<X1Title wordWrapEnabled="false" />
<Y1MajorTick visible="true" />
<Y1TickLabel horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/> </Y1TickLabel>
<Y1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/> </Y1Title>
<Y2TickLabel horizontalAlignment="RIGHT" />
<Y2Title wordWrapEnabled="false" />
<Y1ReferenceLine>
<ReferenceLine index="0" lineWidth="2" displayedInLegend="true" lineColor==⇒
"#ff0000" /> </Y1ReferenceLine>
</Graph>

```

combo_basic.xml

```

<?xml version="1.0" ?>
<Graph version="3.2.0.22" seriesTooltipLabelType="TLT_NONE"
groupTooltipLabelType==⇒
"TLT_NONE" markerTooltipType="MTT_VALUES" markerDisplayed="true" markerShape⇒
InLegend="true" graphicAntialiasing="true" textAntialiasing="true" seriesEffect==⇒
"SE_AUTO_GRADIENT" frameSizeAutomatic="false">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</Footnote>
<LegendArea visible="true" automaticPlacement="AP_NEVER" position="LAP_TOP" fill⇒
Color="#ffffffff" />
<LegendText horizontalAlignment="LEFT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/> </LegendText>
<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_AUTOMATIC" tickLabel⇒
SkipCount="0" tickLabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒

```

```

        />
    </O1TickLabel>
    <O1Title>
        <GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
        />
    </O1Title>
    <SeriesItems defaultMarkerShape="MS_NONE" defaultFitlineType="FT_NONE">
        <Series id="0" markerType="MT_BAR" />
        <Series id="1" markerType="MT_MARKER" markerShape="MS_DIAMOND" lineWidth="2"
color⇒
        ⇒
        ⇒
        ⇒
        ⇒
        ⇒
        ⇒
        ⇒
        ⇒
        ⇒
        "#ff0000" />
        <Series id="2" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="3" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="4" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="5" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="6" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="7" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="8" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="9" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="10" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="11" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="12" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="13" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="14" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="15" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="16" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="17" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="18" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="19" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="20" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="21" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="22" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="23" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="24" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="25" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="26" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="27" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="28" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="29" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
    </SeriesItems>
    <Y1MajorTick visible="true" />
    <Y1TickLabel>
        <GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
        />
    </Y1TickLabel>
    <Y1Title>
        <GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒

```

```

        />
    </Y1Title>
    <Y2MajorTick visible="true" />
    <Y1ReferenceLine>
        <ReferenceLine index="0" lineWidth="2" lineColor="#ff0000" displayedInLegend=>
            "false" />
    </Y1ReferenceLine>
</Graph>
```

combo_markers.xml

```

<?xml version="1.0" ?>
<Graph version="3.2.0.22" seriesTooltipLabelType="TLT_NONE"
groupTooltipLabelType=>
    "TLT_NONE" markerTooltipType="MTT_VALUES" markerDisplayed="true" markerShape=>
    InLegend="true" graphicAntialiasing="true" textAntialiasing="true" seriesEffect=>
    "SE_AUTO_GRADIENT" frameSizeAutomatic="false">
    <Background fillColor="#f1f6f0" />
    <Footnote horizontalAlignment="RIGHT">
        <GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"=>
            />
    </Footnote>
    <LegendArea visible="true" automaticPlacement="AP_NEVER" position="LAP_TOP" fill=>
    Color="#ffffffff"/>
    <LegendText horizontalAlignment="LEFT">
        <GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"=>
            />
    </LegendText>
    <O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_AUTOMATIC" tickLabel=>
    SkipCount="0" tickLabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
        <GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"=>
            />
    </O1TickLabel>
    <O1Title>
        <GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"=>
            />
    </O1Title>
    <SeriesItems defaultMarkerShape="MS_NONE" defaultFitlineType="FT_NONE">
        <Series id="0" markerType="MT_BAR" />
        <Series id="1" markerType="MT_MARKER" markerShape="MS_DIAMOND" lineWidth="2"
color=>
            =>
            =>
            =>
            =>
            =>
            =>
            =>
            "#ff0000" />
        <Series id="2" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="3" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="4" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="5" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
        <Series id="6" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
```

```

<Series id="7" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="8" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="9" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="10" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="11" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="12" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="13" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="14" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="15" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="16" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="17" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="18" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="19" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="20" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="21" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="22" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="23" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="24" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="25" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="26" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="27" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="28" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
<Series id="29" markerType="MT_MARKER" markerShape="MS_DIAMOND" />
</SeriesItems>
<Y1MajorTick visible="true" />
<Y1TickLabel>
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/
</Y1TickLabel>
<Y1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/
</Y1Title>
<Y2MajorTick visible="true" />
<Y1ReferenceLine>
<ReferenceLine index="0" lineWidth="2" lineColor="#ff0000" displayedInLegend⇒
"false" />
</Y1ReferenceLine>
</Graph>

```

line_basic.xml

```

<?xml version="1.0" ?>
<Graph version="3.2.0.22" graphType="LINE_VERT_ABS" depthAngle="0"
legendScrollbar⇒
Presence="SP_NEVER" legendSeriesCount="0" graphicAntialiasing="true" text⇒
Antialiasing="true" markerShapeInLegend="true" frameSizeAutomatic="false">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/
</Footnote>
<LegendArea visible="true" automaticPlacement="AP_NEVER" position="LAP_TOP" fill⇒
Color="#ffffffff"/>
<LegendText horizontalAlignment="LEFT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/

```

```

</LegendText>
<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_AUTOMATIC" tickLabel⇒
SkipCount="0" tickLabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</O1TickLabel>
<O1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</O1Title>
<Y1MajorTick visible="true" />
<Y1TickLabel>
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</Y1TickLabel>
<Y1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</Y1Title>
<Y2MajorTick visible="true" />
</Graph>

```

pie_basic.xml

```

<?xml version="1.0" ?>
<Graph version="3.2.0.22" autoLayout="AL_ALWAYS" depthAngle="50" depthRadius="0"⇒
pieDepth="0" groupTooltipLabelType="TLT_NONE" graphicAntialiasing="true" text⇒
Antialiasing="true" pieTilt="0" seriesEffect="SE_AUTO_GRADIENT" graphType="PIE">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</Footnote>
<LegendArea visible="false" />
<LegendText horizontalAlignment="CENTER" />
<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_NOSKIP" automatic⇒
Rotation="AR_NO_ROTATE" />
<O1Title wordWrapEnabled="false" />
<PlotArea fillColor="#f1f6f0" />
<PieLabel>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</PieLabel>
<Slice labelPosition="LP_OUTSIDE_WITH_FEELER" />
<SliceLabel textType="LD_TEXT_PERCENT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</SliceLabel>
<X1Title wordWrapEnabled="false" />
<Y1TickLabel horizontalAlignment="RIGHT" />
<Y1Title wordWrapEnabled="false" />
<Y2TickLabel horizontalAlignment="RIGHT" />
<Y2Title wordWrapEnabled="false" />
</Graph>

```

pie_onetime.xml

```
<?xml version="1.0" ?>
<Graph version="3.2.0.22" autoLayout="AL_ALWAYS" depthAngle="50" depthRadius="0"⇒
 pieDepth="0" groupTooltipLabelType="TLT_NONE" graphicAntialiasing="true" text⇒
 Antialiasing="true" pieTilt="0" seriesEffect="SE_AUTO_GRADIENT" graphType="PIE">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"⇒
 underline="false"⇒
 />
</Footnote>
<LegendArea visible="false" />
<LegendText horizontalAlignment="CENTER" />
<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_NOSKIP" automatic⇒
 Rotation="AR_NO_ROTATE" />
<O1Title wordWrapEnabled="false" />
<PlotArea fillColor="#f1f6f0" />
<PieLabel>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"⇒
 underline="false"⇒
 />
</PieLabel>
<SeriesItems>
<Series id="0" color="#336699" />
<Series id="1" color="#ffff00" />
<Series id="2" color="#ff0000" />
</SeriesItems>
<Slice labelPosition="LP_OUTSIDE_WITH_FEELER" />
<SliceLabel textType="LD_TEXT_PERCENT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"⇒
 underline="false"⇒
 />
</SliceLabel>
<X1Title wordWrapEnabled="false" />
<Y1TickLabel horizontalAlignment="RIGHT" />
<Y1Title wordWrapEnabled="false" />
<Y2TickLabel horizontalAlignment="RIGHT" />
<Y2Title wordWrapEnabled="false" />
</Graph>
```

stacked_bar_basic.xml

```
<?xml version="1.0" ?>
<Graph version="3.2.0.22" seriesTooltipLabelType="TLT_NONE"⇒
 groupTooltipLabelType=>
 "TLT_NONE" markerTooltipType="MTT_VALUES" graphicAntialiasing="true" text⇒
 Antialiasing="true" seriesEffect="SE_AUTO_GRADIENT" graphType="BAR_VERT_STACK">
<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"⇒
 underline="false"⇒
 />
</Footnote>
<LegendArea automaticPlacement="AP_NEVER" position="LAP_TOP" fillColor="#ffffff" ⇒
 /> <LegendText horizontalAlignment="LEFT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"⇒
 underline="false"⇒
 />
</LegendText>
```

```

<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_AUTOMATIC" tickLabel⇒
SkipCount="0" tickLabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</O1TickLabel>
<O1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</O1Title>
<Y1ReferenceLine>
<ReferenceLine index="0" lineWidth="2" lineColor="#ff0000" displayedInLegend⇒
"true" />
</Y1ReferenceLine>
<Y1TickLabel>
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</Y1TickLabel>
<Y1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</Y1Title>
</Graph>

```

stacked_bar_ontime.xml

```

<?xml version="1.0" ?>

<Graph version="3.2.0.22" seriesTooltipLabelType="TLT_NONE"
groupTooltipLabelType=>
"TLT_NONE" markerTooltipType="MTT_PERCENT_VAL" graphicAntialiasing="true"⇒
textAntialiasing="true" seriesEffect="SE_AUTO_GRADIENT" graphType="BAR_VERT_⇒
PERCENT">

<Background fillColor="#f1f6f0" />
<Footnote horizontalAlignment="RIGHT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</Footnote>
<LegendArea automaticPlacement="AP_NEVER" position="LAP_TOP" fillColor="#ffffff" ⇒
/>
<LegendText horizontalAlignment="LEFT">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</LegendText>
<O1TickLabel textRotation="TR_HORIZ" tickLabelSkipMode="TLS_AUTOMATIC" tickLabel⇒
SkipCount="0" tickLabelSkipFirst="0" automaticRotation="AR_HORIZ_ROTATE_270">
<GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"⇒
/>
</O1TickLabel>
<O1Title>
<GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"⇒
/>
</O1Title>

```

```

<SeriesItems>
  <Series id="0" color="#336699" />
  <Series id="1" color="#ffff00" />
  <Series id="2" color="#ff0000" />
</SeriesItems>
<Y1ReferenceLine>
  <ReferenceLine index="0" lineWidth="2" lineColor="#ff0000" displayedInLegend=>
    "true" />
</Y1ReferenceLine>
<Y1TickLabel>
  <GraphFont name="SansSerif" size="9" bold="false" italic="false"
underline="false"=>
    />
</Y1TickLabel>
<Y1Title>
  <GraphFont name="SansSerif" size="11" bold="true" italic="false"
underline="false"=>
    />
</Y1Title>
</Graph>

```

3.3.7 Example Graph DTD

This section contains example graph DTD for each of the chart type provided in EnterpriseOne. You can modify the code in these graph DTDs to alter how the chart displays information.

```

<?xml version='1.0' encoding="UTF-8"?>

<!ELEMENT Graph
(Footnote?,MarkerText?,O1Title?,Title?,Subtitle?,X1Axis?,Y1Axis?,Y1Reference=>
Line?,Y1Title?,Y2Axis?,Y2Title?,LocalRelationalData)
<!--the graphName is the name of a pre-defined template to use--&gt;
&lt;!ATTLIST Graph
  graphName CDATA #REQUIRED
<!-- Graph definition of the footnote --&gt;
&lt;!ELEMENT Footnote EMPTY&gt;
&lt;!ATTLIST Footnote
  text CDATA #IMPLIED
  visible (true | false) #IMPLIED
&lt;!ELEMENT MarkerText (GraphFont?, X1ViewFormat?, Y1ViewFormat?, Y2ViewFormat?, =&gt;
ZViewFormat?) &gt;
<!-- MarkerText attributes:
  markerTextPlace: where, in relation to the marker, the marker text appears;
    outside max is above bar
    on max edge is sitting on the bar
    inside max is on the bar, near the top
    inside min is on the bar, near the bottom
    center is centered on the bar
    custom lets you set the angle and radius
  **next to attributes in effect only when markerTextPlace is MTP_CUSTOM**
  markerTextAngleDefault: angle of the text, in degrees, measured from the
    center of the marker
  markerTextRadiusDefault: distance from the center of the marker, 0 - 100
--&gt;
&lt;!ATTLIST MarkerText
  visible (true | false) #IMPLIED
  fontTypeAbsolute (true | false) #IMPLIED
</pre>

```

```

markerTextPlace (MTP_OUTSIDE_MAX | MTP_ON_MAXEDGE | MTP_INSIDE_MAX |
                 MTP_INSIDE_MIN | MTP_CENTER | MTP_CUSTOM) #IMPLIED
markerTextAngleDefault CDATA #IMPLIED
markerTextRadiusDefault CDATA #IMPLIED >

<!-- Title for the ordinal axis (O1Axis)-->
<!ELEMENT O1Title EMPTY>
<!ATTLIST O1Title
    text CDATA #IMPLIED
    visible (true | false) #IMPLIED
>
<!-- Graph definition of the title; appears in graph image -->
<!ELEMENT Title EMPTY>
<!ATTLIST Title
    text CDATA #IMPLIED
    visible (true | false) #IMPLIED
>
<!-- Graph definition of the subtitle; appears in graph image -->
<!ELEMENT Subtitle EMPTY>
<!ATTLIST Subtitle
    text CDATA #IMPLIED
    visible (true | false) #IMPLIED
>
<!-- Reference line for Y1-axis: includes up to 3 actual lines -->
<!ELEMENT Y1ReferenceLine (ReferenceLine*) >

<!-- Represents the actual line in ReferenceLine element -->
<!ELEMENT ReferenceLine EMPTY >

<!-- ReferenceLine attributes:
    index: which line in this axis; 0, 1, or 2
    text: description of reference line
    value: where along the axis the line should appear
    displayedInLegend: whether a key to the line appears in the legend
-->
<!ATTLIST ReferenceLine
    index CDATA #REQUIRED
    visible (true | false) #IMPLIED
    lineWidth CDATA #IMPLIED
    text CDATA #IMPLIED
    value CDATA #IMPLIED
    displayedInLegend (true | false) #IMPLIED
    lineColor CDATA #IMPLIED
    lineStyle (LS_SOLID | LS_DASH | LS_DOTTED | LS_DASH_DOT) #IMPLIED >
<!-- Title for the Y1-axis (left side); reads up -->
<!ELEMENT Y1Title EMPTY>
<!ATTLIST Y1Title
    text CDATA #IMPLIED
    visible (true | false) #IMPLIED
>

<!-- Title for the Y2-axis (right side); reads down -->
<!ELEMENT Y2Title EMPTY>
<!ATTLIST Y2Title
    text CDATA #IMPLIED
    visible (true | false) #IMPLIED
>
<!-- Relational data for local data source:
    Graph builds a grid of data from this element and children -->
<!ELEMENT LocalRelationalData

```

```

(Row+)
>
<!ELEMENT Row EMPTY>
<!ATTLIST Row
columnKey CDATA #IMPLIED
dataValue CDATA #IMPLIED
rowKey CDATA #IMPLIED
>
<!-- ViewFormat specifies number formatting-->
<!ELEMENT ViewFormat EMPTY >
<!-- ViewFormat attributes
numberType: format numbers as currency or percent?
thousandSeparator: character for separating thousands
decimalSeparator: character for separating whole number from decimal
currencySymbol: currency symbol; numberType must be NUMTYPE_CURRENCY
leadingZero: display leading zeros?
decimalDigit: number of decimal digits to display
posNumFmt: format for positive numbers
negNumFmt: format for negative numbers
posCurFmt: format for positive currency
negCurFmt: format for negative currency
scaleFactor: how much to scale (abbreviate) numbers
scaleDownThousands: character(s) to use when numbers represent thousands
scaleDownMillions: character(s) to use when numbers represent millions
scaleDownBillions: character(s) to use when numbers represent billions
scaleDownTrillions: character(s) to use when numbers represent trillions
scaleDownQuadrillions: character(s) to use when numbers represent
quadrillions
javaDateFormat: Java date format
oracleDateFormat: Oracle date format

"used" attributes specify whether one of the above properties has
been explicitly set; used for merging when using rules
-->
<!ATTLIST ViewFormat
numberType (NUMTYPE_GENERAL | NUMTYPE_CURRENCY | NUMTYPE_PERCENT) #IMPLIED
thousandSeparator CDATA #IMPLIED
decimalSeparator CDATA #IMPLIED
currencySymbol CDATA #IMPLIED
leadingZero (true | false) #IMPLIED
decimalDigit CDATA #IMPLIED
posNumFmt (POS_NUMFMT_NUM | POS_NUMFMT_POS_NUM | POS_NUMFMT_NUM_POS ) #IMPLIED
negNumFmt (NEG_NUMFMT_OP_NUM_CP | NEG_NUMFMT_NEG_NUM | NEG_NUMFMT_NUM_NEG )#IMPLIED
#IMPLIED
posCurFmt (POS_CURFMT_CUR_NUM | POS_CURFMT_NUM_CUR |
POS_CURFMT_CUR_SPAC_NUM | POS_CURFMT_NUM_SPAC_CUR |
POS_CURFMT_POS_CUR_NUM | POS_CURFMT_CUR_NUM_POS |
POS_CURFMT_POS_NUM_CUR | POS_CURFMT_NUM_POS_CUR |
POS_CURFMT_NUM_CUR_POS | POS_CURFMT_POS_NUM_SPAC_CUR |
POS_CURFMT_POS_CUR_SPAC_NUM | POS_CURFMT_NUM_SPAC_CUR_POS )
#IMPLIED
negCurFmt (NEG_CURFMT_OP_CUR_NUM_CP | NEG_CURFMT_NEG_CUR_NUM |
NEG_CURFMT_CUR_NEG_NUM | NEG_CURFMT_CUR_NUM_NEG |
NEG_CURFMT_OP_NUM_CUR_CP | NEG_CURFMT_NEG_NUM_CUR |
NEG_CURFMT_NUM_NEG_CUR | NEG_CURFMT_NUM_CUR_NEG |
NEG_CURFMT_NEG_NUM_SPAC_CUR | NEG_CURFMT_NEG_CUR_SPAC_NUM |
NEG_CURFMT_NUM_SPAC_CUR_NEG ) #IMPLIED
scaleFactor (SCALEFACTOR_NONE | SCALEFACTOR_THOUSANDS |
SCALEFACTOR_MILLIONS | SCALEFACTOR_BILLIONS |
SCALEFACTOR_TRILLIONS | SCALEFACTOR_QUADRILLIONS ) #IMPLIED

```

```
scaleDownThousands CDATA #IMPLIED
scaleDownMillions CDATA #IMPLIED
scaleDownBillions CDATA #IMPLIED
scaleDownTrillions CDATA #IMPLIED
scaleDownQuadrillions CDATA #IMPLIED
javaDateFormat CDATA #IMPLIED
oracleDateFormat CDATA #IMPLIED
numberTypeUsed (true | false) #IMPLIED
thousandSeparatorUsed (true | false) #IMPLIED
decimalSeparatorUsed (true | false) #IMPLIED
currencySymbolUsed (true | false) #IMPLIED
leadingZeroUsed (true | false) #IMPLIED
decimalDigitUsed (true | false) #IMPLIED
posNumFmtUsed (true | false) #IMPLIED
negNumFmtUsed (true | false) #IMPLIED
posCurFmtUsed (true | false) #IMPLIED
negCurFmtUsed (true | false) #IMPLIED
scaleFactorUsed (true | false) #IMPLIED
scaleDownThousandsUsed (true | false) #IMPLIED
scaleDownMillionsUsed (true | false) #IMPLIED
scaleDownBillionsUsed (true | false) #IMPLIED
scaleDownTrillionsUsed (true | false) #IMPLIED
scaleDownQuadrillionsUsed (true | false) #IMPLIED
javaDateFormatUsed (true | false) #IMPLIED
oracleDateFormatUsed (true | false) #IMPLIED >

<!-- ViewFormats for different number values that can appear in marker
     text and tooltip text -->
<!ELEMENT X1Axis (ViewFormat?) >
<!ELEMENT X1ViewFormat (ViewFormat) >
<!ELEMENT Y1Axis (ViewFormat?) >
<!ELEMENT Y1ViewFormat (ViewFormat) >
<!ELEMENT Y2Axis (ViewFormat?) >
<!ELEMENT Y2ViewFormat (ViewFormat) >
<!ELEMENT ZViewFormat (ViewFormat) >
```

3.4 Attaching Data Items to a Control

This section provides an overview of the relationship between data items and controls and discusses how to:

- Attach a data item to a control.
- Override a default DD trigger.

3.4.1 Understanding the Relationship between Data Dictionary Items and Controls

On a form, you can use any data item that is in the BV and any data item from the DD. BV items are associated with the database; they are retrieved from and updated to the database tables. DD items are not connected to the database; their values are not retrieved from nor updated to the database tables by the system. On a form, BV items appear with a blue box in the left corner, and DD items appear with a yellow box in the left corner. Use DD items as work fields for ER or fields that are not directly connected to the database table.

You can associate data item values with these control types:

- Check boxes.

- Edit controls.
- Grid controls (columns).
- Parent child controls (columns).
- Radio buttons.

Radio buttons must be associated with DD items. All radio buttons with the same DD item are considered to be a group. Selecting one radio button within the group automatically clears all other radio buttons in the same group.

- Combo boxes

The rest of the controls types either cannot be associated with a data item, such as push buttons, or essentially act as containers for controls, such as subforms and tab pages, which may in turn be associated with a data item, according to their type.

If you want the user-entered value for a control to update a database record, then the control must be associated with a BV item. For example, a check box can be associated with the database field called Taxable. In the check box properties, the Checked value should be Y and the Unchecked value should be N. You can use these values in ER as well.

Each DD item has a set of properties (such as Default Value) and behaviors (such as Format Rules and Edit Rules). When you create a DD item, you define its default properties and behaviors at that time. You can override previously-defined DD items at the application level, which enables you to further customize how data items behave at runtime. You can disable default properties and behaviors for a specific item, as well as override to use different values and triggers.

3.4.2 Attaching a Data Item to a Control

To attach a data item to a control:

1. Select the control, and then click Data Item Information in the Property Browser and click the ellipses button that appears in the Data Item Information field.
2. On Control Properties, perform one of these tasks:
 - To attach a data item from the BV attached to the form, click the Business View Items tab and double-click a data item.
 - To attach a data item from the DD, click the Data Items tab, search for the data item you want, and then double-click it.
3. To override the data item name as it appears on the form, click the General tab, click Override Text, and then enter the name you want to use in the Event Rules Title field.

The Text is Overridden property in the Property Browser changes to Yes. To return the display name to the DD name, change Text is Overridden to No.

3.4.3 Overriding a Default Data Dictionary Trigger

To override default DD properties and triggers.

1. On the form, double-click a control with an attached DD item.
2. On Edit Properties, click the Overrides tab, and then click Data Dictionary Overrides.
3. On Data Dictionary Overrides, to disable one or more triggers, select the ones you want to disable in the Disable box.

4. To override a trigger, click the applicable trigger type and complete the form that appears.

For example, if you click Default Value, the Override Default Value form appears and you would make your changes there.

3.5 Associating a Data Item Description with a Field

This section provides an overview of the relationship between data item descriptions and fields and discusses how to display the title of a data item associated with a field.

3.5.1 Understanding the Relationship between Data Item Descriptions and Fields

When you insert an edit control on the form and associate a data item with it, FDA does not automatically associate a description with the field. If you want to display the row description for the data item automatically on the form, you must associate the data item description with the field.

When you associate a description with an edit control, a description of the value in the control automatically appears next to it. Associating a description is optional but is useful as a visual aid on the form.

For example, suppose that the address book number, for example, 1001 appears in an edit control or in an edit control. When the user presses Tab to move out of the control, the address book name, XYZ Company, appears next to the control as the associated description.

3.5.2 Displaying the Title of a Data Item Associated with a Field

To display the title of a data item associated with a field:

1. Click the control, and then select Edit, Associate Description.

FDA creates a static text block and attaches it to the cursor so that you can place it where you want.

2. Click the approximate location on the form where you want the description to appear.

3.6 Grouping Controls

Sometimes, you want a set of controls to be taken as a whole by the user. The classic example is when you want the user to select one item from a list of items, and this situation is typically resolved by creating a radio button for each option and then grouping them so that the user can select only one at a time.

To provide the user with a visual cue that the controls are grouped, you can place a group box around the controls. Use the Title property for the group box to label the group on the form.

If you want to create more space on a form or only display controls when needed, you can apply the Collapsible property. When the Collapsible property has been applied to the group box a title bar will appear. Clicking on the title bar will expand / collapse the group box. Once the Collapsible property is applied you can also use the Expand Group Box and Collapse Group Box Control System Functions within Event Rules. You can place a collapsible group box next to another (parallel) as long as the group box property Position -Top is the same value for both.

3.7 Setting the Tab Sequence of Controls on a Form

This section provides an overview of tab sequences and discusses how to change the tab sequence on a form.

3.7.1 Understanding Tab Sequences

The tab sequence of the controls determines the order in which the cursor travels through the controls on the form. Each control that is designated as a tab stop is numbered to reflect how the cursor will travel. For example, when the user opens the form, the cursor appears on the control labeled number one. When the user presses Tab, the cursor moves sequentially to the control (tab stop) labeled number two on the form.

Only those controls that are designated as tab stops in the control properties are affected by the tab sequence. The default tab sequence is the order in which you place the controls on the form.

These properties can override the first tab stop:

- Default cursor on add mode
- Default cursor on update mode

3.7.2 Changing the Tab Sequence on a Form

To change the tab sequence:

1. On the form with which you are working, select Layout, and then Tab Sequence. FDA displays each control so you see its number in the tab sequence instead of its name.
2. Click the controls in the order in which you want the cursor to travel.

The first control that you click becomes number one, the second becomes number two, and so on. To reset the numbers to their original values, right-click anywhere in the window.

Note: Subforms contain their own sequence orders for objects on the subform. Therefore, you set the tab sequence on each subform within the subform itself. The subform, then, becomes a single object in the tab sequence for the power form.

3. Click Layout, and then Tab Sequence again to stop selecting the sequence.

OR

1. Click Layout, and then click Update Tab Sequence.
2. Right-click on a control on which you want to change the tab sequence. The Tab Sequence field displays.
3. Change the number in the field to reflect where you want that field in sequence to the other fields.
4. Press Enter.

FDA automatically adjusts the remaining fields in the sequence. For example, if you change a field sequence from 2 to 4 the remaining fields adjust as follows:

Original Field Sequence	Adjusted Field Sequence
Field 1	Field 1
Field 2	Field 4
Field 3	Field 2
Field 4	Field 3

3.8 JD Edwards EnterpriseOne FDA Compare

The FDA Compare tool in Form Design Aid (FDA) enables you to compare one version of an application to another. You can compare them on the application level to determine whether forms have been added, deleted, or rearranged and whether the properties have changed. You also can compare the forms in the applications to each other to see whether controls have been added, deleted, or rearranged and whether the properties have changed.

Additionally, you can compare two different applications as well. This feature is useful when you have made a new application by copying an existing one and then modifying it. Then, when you upgrade, you can not only compare the base application to its new counterpart, but also you can compare a custom application.

See "Understanding FDA Compare" in the *JD Edwards EnterpriseOne Tools Software Updates Guide*.

4

Working with Transaction Processing

This chapter provides an overview of transaction processing and discusses how to work with transaction processing.

This chapter contains the following topics:

- [Section 4.1, "Understanding Transaction Processing"](#)
- [Section 4.2, "Implementing Transaction Processing"](#)

4.1 Understanding Transaction Processing

A transaction is a logical unit of work (comprising 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 are made to both tables, or no updates are 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 transaction are:

- Atomic
 - Either all database changes for an entire transaction are completed or none of the changes are completed.
- Consistent
 - Database changes transform from one consistent database state to another.
- Isolated
 - Transactions from concurrent applications do not interfere with each other. The updates from a transaction are not visible to other transactions that execute concurrently until the transaction commits.
- Durable
 - Complete database operations are permanently written to the database.

4.1.1 Commits and Rollbacks

The scope of a transaction is defined by the beginning and end of the transaction. The end of a transaction occurs when the transaction is committed or rolled back. If a transaction is started but not committed or rolled back, the system will roll back the transaction when user exists the system.

Transaction processing uses commits to control database operations. Commits are commands to the database. You can configure transactions to be automatically or manually committed. An auto commit transaction writes database changes permanently immediately when the changes occur. A manual commit transaction will buffer database changes when they occur, and only write database changes permanently when the transaction is committed.

4.1.1.1 Commit

A commit is an explicit command to the database to permanently store the results of operations performed by a statement. This event successfully ends a transaction.

4.1.1.2 Rollback

A rollback is an explicit command to the database to cancel the results of operations performed by a statement. This event indicates that a transaction ended unsuccessfully.

Any failure to insert, update, or delete within a transaction boundary causes all record activity within that transaction to roll back. If no failures have occurred at the end of the transaction, a commit is done, and the records become available to other processes.

In the case of a catastrophic failure (such as due to network problems), the Database Management System (DBMS) performs an automatic rollback. Likewise, if the user clicks Cancel on a form, a rollback command can be issued through a system function.

4.1.2 Transaction Processing

A JD Edwards EnterpriseOne software transaction is a logical unit of work (comprised of one or more SQL statements) performed on any number of databases. A single-statement transaction consists of one statement; a multiple-statement transaction consists of more than one statement.

You can construct a transaction within an application to group multiple database operations. The application can then request the DBMS to buffer the database operations until the application executes a specific command to perform the updates requested within the transaction. Database operations that are not part of a transaction update the database immediately.

If an application has transaction processing turned on, other users cannot see the updated records until an update has been committed. Only processes within that transaction can access records in the transaction until the transaction is complete.

Transaction processing is supported to both interactive applications and batch applications (also called reports). Using Form Design Aid, you can enable transactions for JD Edwards EnterpriseOne forms. You can also design the database operations that are included in a transaction. Using Oracle's JD Edwards EnterpriseOne Report Design Aid, you can enable transactions and design transaction operations for batch applications. Not all applications enable transaction processing. Decide carefully whether transaction processing should be enabled.

If transaction processing is turned on for database operations for tables that reside in DB2 for IBM i, then those tables must be journaled. Journaling can decrease

performance because of the additional processing required. Contact your DB2 administrator if you have problems with the process.

General messages and errors for transaction processing are written in jde.log, jddebug.log or jas logs.

4.1.2.1 Data Interdependence

Data interdependence refers to the situation where data in multiple tables are dependent on each other. For example, a voucher has records in both the F0411 and the F0911 tables. Because data interdependence exists between these two tables, the transaction is incomplete when data is written in one table but not the other.

4.1.2.2 Transaction Boundaries

A transaction boundary encompasses all of the database operations that comprise a transaction. In interactive applications, a transaction boundary might include only the database operations on a single form. When a transaction includes data operations from another form, the transaction boundary must be extended to include that form.

4.1.2.3 Interactive Application Transaction Processing Scenarios

The typical flow for a transaction is:

1. An application starts and calls the runtime engine.
2. The runtime engine initializes the transaction.
3. The runtime engine opens a view.
4. The runtime engine performs database operations.
5. The runtime engine commits database operations.

To include two connected forms in the same transaction boundary, you must activate transaction processing for the parent form and designate *Include in parent* on the interconnection to the second form. You do not need to activate transaction processing for the second form because your choice on the originating form overrides your choice on the called form.

This table outlines the relationship between two forms and the transaction boundaries that exist in each scenario. Transactions can be started in one form and extended to more forms through form interconnects and business function calls. In the example, the OK button on Form 1 invokes Form 2. You can change the transaction boundaries by specifying TP On or TP Off. The table explains what happens when you define your transaction boundary in various ways.

Scenario	Forms	TP On	TP Off	"Include in Parent" flag On for Form Interconnect, BSFN, Table I/O	Comments
A	Form 1 Form 2		X X		All forms use Auto Commit.

Scenario	Forms	"Include in Parent" flag On for Form Interconnect, BSFN, Table I/O			Comments
		TP On	TP Off		
B	Form 1		X	X	Because neither form uses Manual Commit, the Include in Parent flag on Form Interconnect Properties is ignored.
	Form 2		X		All forms use Auto Commit
C	Form 1	X	O		Form 1 (parent) uses Manual Commit mode, and Form 2 (child) uses Auto Commit.
	Form 2	O	X		Because the Include in Parent flag is Off, the transaction boundary does not extend to include Form 2 (child)
D	Form 1	X	O	X	Even though the transaction processing flag is Off for Form 2 (child), the Include in Parent flag is On.
	Form 2	O	X		The transaction boundary extends to include Form 2 (child)
E	Form 1	O	X		Because the Include in Parent flag is Off, Form 1 (parent) and Form 2 (child) operate as independent entities.
	Form 2	X	O		Form 1 operates in Auto Commit mode and Form 2 operates in Manual Commit mode.
F	Form 1	O	X	X	An atypical case. Because transaction processing is Off on Form 1 (parent), the transaction boundary does not extend to the child, even though the Include in Parent flag is On for Form 2 (child)
	Form 2	X	O		Form 2 (child) is in Manual Commit mode and the interconnect flag is ignored.

Scenario	Forms	TP On	TP Off	"Include in Parent" flag On for Form Interconnect, BSFN, Table I/O	Comments
G	Form 1 Form 2	X X			Transaction processing is On for both forms. Because in Include in Parent flag is Off, each form is a transaction boundary and a commit is issued for each.
H	Form 1 Form 2	X X	X		Transaction processing is On for both forms. However, because the Include in Parent flag is On, one transaction will incorporate both forms. The transaction boundary encompasses both forms. Form 2 is a child of Form 1.

4.1.2.4 Transaction Processing and Business Functions

An application or batch process establishes the primary transaction boundary. If an application calls a business function in Event Rules, the database operations in the business function are grouped within the boundaries of their parent application.

Master business functions should not define their own boundaries.

If your application calls several business functions, and you need to include the business functions in the transaction boundary, you must enable transaction processing for the application. Should a failure occur and you need to roll back database operations for the business function, you must designate Include in Transaction on the business function call.

Note: When you use business functions within a transaction, you must be careful not to cause a deadlock. If two functions modify the same table, you might cause a deadlock if you include one function in the transaction but not the other. On the other hand, if a business function that selects records for information also updates or inserts data in other tables, you might want to split the business function.

4.1.2.5 Transaction Processing in Remote Business Functions

In a transaction-enabled application, if a server business function has modified a record, and a client business function outside the transaction attempts to access the record, the client function is locked out until the server business function has committed the data. Until the data is committed, the client application cannot access database changes performed by server-side business functions. If a server business function fails to commit or a user cancels a transaction on a server business function, the transaction for the business function rolls back.

4.1.3 Transaction Processing Availability

For interactive applications, transaction processing is available for these form types:

- Fix/Inspect
- Header detail
- Headerless detail
- Power Edit
- Editable Subforms
- Wizard Form
- Edit Portlet

Transaction processing is only available during OK processing for these events:

- OK Post Button Clicked - Asynch
- OK Post Button Clicked
- Add Record to DB – Before
- Add Record to DB – After
- Update Record to DB – Before
- Update Record to DB – After
- Add Grid Rec to DB – Before
- Add Grid Rec to DB – After
- All Grid Recs Added to DB
- Update Grid Rec to DB – Before
- Update Grid Rec to DB – After
- All Grid Recs Updated to DB
- Delete Grid Rec from DB – Before
- Delete Grid Rec from DB – After
- All Grid Recs Deleted from DB

Actions that occur outside of these events are not within the transaction boundary.

Note: Transaction processing is also available for reports and batch applications. You can use the transaction processing system functions to define transaction boundary. You can also extend transaction boundaries through business function calls and table I/O calls.

4.2 Implementing Transaction Processing

This section discusses how to:

- Define transaction processing for a form.
- Extend a transaction boundary.

4.2.1 Forms Used to Work with Transaction Processing in Interactive Applications

Form Name	FormID	Navigation	Usage
Form Design Aid	N/A	On Solution Explorer, type OMW in the Fast Path field. On Object Management Workbench, select an interactive application and click the Design Tools button. On the Interactive Applications Design form, select the Design Tools tab, and then click Start Form Design Aid button.	Create and modify interactive applications.

4.2.2 Defining Transaction Processing for a Form

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

If a transaction includes a single form, then this is the only action that is required because the form itself is the transaction boundary. However, if the transaction included data from another form, then you must extend the boundary to the applicable form through a form interconnection.

Note: You can also extend transaction boundaries using business functions or table I/O.

To define transaction processing for a form:

Access Form Design.

1. Double click the form for which you want to access Form Properties.
2. Click the Transaction Properties style option.

4.2.3 Extending a Transaction Boundary

You can extend the transaction boundary from one form to another form by setting up a parent/child relationship between the forms. To extend the boundary enable the Transaction Processing flag through a form interconnection in Event Rules Design.

In Form Design, define form properties so that each form within the transaction boundary includes transaction processing.

4.2.3.1 Extending a Transaction Boundary between Forms

If a parent form uses manual commit, the form to which you connect it must also use manual commit.

Access Form Design Aid.

1. Select the parent form with which you are working.
2. From the Form menu, select Menu, Toolbar Exits.

3. Select the OK row and click the Event Rules button.
4. From Event Rules Design, select the Button Clicked event, and click the Form Interconnect button.
5. On Work with Applications, select the application that you want to use.
6. On Work with Forms, select the form that you want to include in the transaction boundary.
7. On Work with Versions, select the version of the application that you want to use.
8. On Form Interconnect, click the Include in Transaction Transaction Processing option, and click OK.

4.2.3.2 Extending a Transaction Boundary by Using Business Functions

You can include a business function in a transaction boundary. If the parent form uses automatic commit, the business function to which you extend the transaction boundary also uses automatic commit. Any business function that is not marked as Include in Transaction uses auto-commit. Business functions that process asynchronously can also be included in a transaction.

Access the Form Design Aid.

1. Select the parent form with which you are working.
2. From the Form menu, select Menu, Toolbar Exits.
3. Select the OK row and click the Event Rules button.
4. From Event Rules Design, select the *Button Clicked* event, and click the Business Functions button.
5. From Business Function Search, select the business function that you want to include in the transaction boundary.
6. On Business Function, click the Include in Transaction Transaction Processing option.

Business Functions marked for both Asynchronous and Include in Transaction are included in the Transaction Boundary.

4.2.3.3 Extending a Transaction Boundary by Using Table I/O

Transaction processing is available only for the Open Table operation in Table I/O. The opening of a table determines whether interaction with the table will be manual commit (part of a transaction) or automatic commit. Any Open Table operation not marked as Include in Transaction uses automatic commit.

Access the Form Design Aid.

1. Select the parent form with which you are working.
2. From the Form menu, select Menu, Toolbar Exits.
3. Select the OK row and click the Event Rules button.
4. From Event Rules Design, select the *Button Clicked* event, and click the Table I/O button.
5. On Insert Table I/O Operation, select the Open option under Advanced Operations, and then click Next.
6. On Data Source, click Advanced Options.
7. On Advanced Options, select the Include in Transaction option, and then click OK.

8. On Data Source, click Finish.

The Open operation appears in your event rules.

4.2.3.4 Defining Transaction Processing for a Report

In addition to interactive transaction processing, JD Edwards EnterpriseOne software also provides transaction processing for reports and batch processes. To enable transaction processing for a batch process, click the Advanced tab for report properties and select Transaction Processing. Then use the Transaction Processing system functions to define the beginning and ending boundaries of the transactions. You can also extend your transaction boundaries to include business functions and table I/O.

Understanding Find/Browse Forms

This chapter contains the following topics:

- [Section 5.1, "Find/Browse Forms"](#)
- [Section 5.2, "Find/Browse Events"](#)
- [Section 5.3, "Find/Browse Runtime Processing"](#)

5.1 Find/Browse Forms

Find/Browse forms are used to query business views (BVs) and to select records from BVs for operations.

5.2 Find/Browse Events

These events can occur on the find/browse form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Write Grid Line-Before
- Write Grid Line-After
- Last Grid Record Has Been Fetched
- Call is Alerting
- End Dialog
- XAPI Subscribe Event

Note: Starting with EnterpriseOne Tools Release 9.1.4, the Form Service Request event is available within FDA for all forms except wizard and message forms. This event is used only within the context of mobile enterprise application development. See "Using the Form Service Request Event in FDA" in the *JD Edwards EnterpriseOne Tools Developing and Customizing Mobile Enterprise Applications Guide*.

5.3 Find/Browse Runtime Processing

This section discusses how runtime processes find/browse forms.

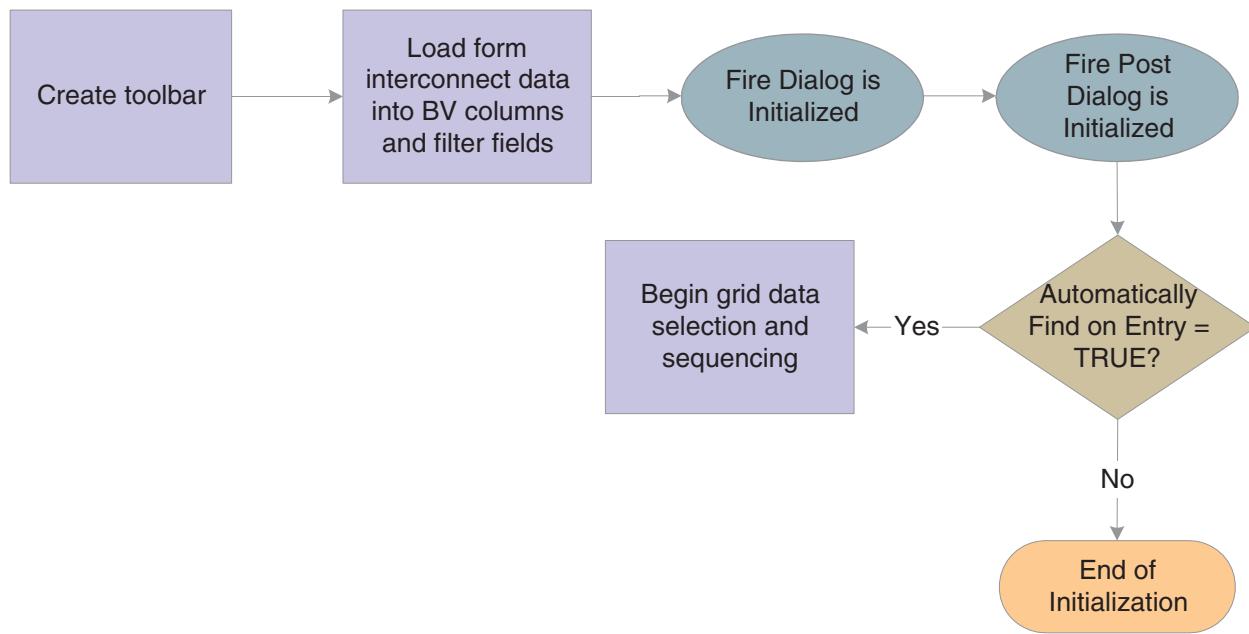
5.3.1 Dialog Initialization

When a find/browse form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. Business view columns (BC)
4. Form controls (FC)
5. Grid fields
6. Static text
7. Helps
8. Event rules structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 5–1 Find/Browse dialog initialization



Grid data selection and sequencing occurs at the control level.

See [Grid Control System Functions](#).

The system creates an internal structure that represents the data selection and data sequencing requirements specified by the user. The system then passes this to the database engine to perform the actual database select and sequencing. The data used for selection is based on values from filter fields and query-by-example (QBE) columns. The system holds the data until the data is retrieved.

5.3.2 Find Button

The Find button is a standard button on find/browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the

filter fields, runtime performs data selection and sequencing for the grid. After reloading the grid with the fetched data, runtime fires the **Post Button Clicked** event.

5.3.3 Select Button

The Select button is a standard button on find/browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors occur, runtime writes the values from the selected row to the BC and fires the **Post Button Clicked** event. Then it fires the **End Dialog** event and initiates the dialog close process.

5.3.4 Close Button

The Close button is a standard button on find/browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime attempts to close all of the modeless child forms, if any exist. If any of these child forms cannot be closed, the Close button process is terminated. Otherwise, runtime fires the **End Dialog** event and initiates the dialog close process.

5.3.5 Dialog Close

Find/Browse can be closed either by the user (typically by clicking the Select or Close buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BC for database commit.
2. Persist saved queries and grid formats
3. Terminate error and thread handling.
4. Terminate helps.
5. Destroy the window.

6

Understanding Fix/Inspect Forms

This chapter contains the following topics:

- [Section 6.1, "Fix/Inspect Forms"](#)
- [Section 6.2, "Fix/Inspect Events"](#)
- [Section 6.3, "Fix/Inspect Runtime Processing"](#)

6.1 Fix/Inspect Forms

A fix/inspect form is used to update and insert single database records. This form type can also be used to display static information to the user, as well as prompt the user for information. Sign-on screens, for example, prompt the user to enter sign-on information.

6.2 Fix/Inspect Events

These events can occur on the fix/inspect form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Clear Screen Before Add
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Post Commit
- **End Dialog**
- **XAPI Subscribe Event**

6.3 Fix/Inspect Runtime Processing

This section discusses how runtime processes fix/inspect forms.

6.3.1 Dialog Initialization

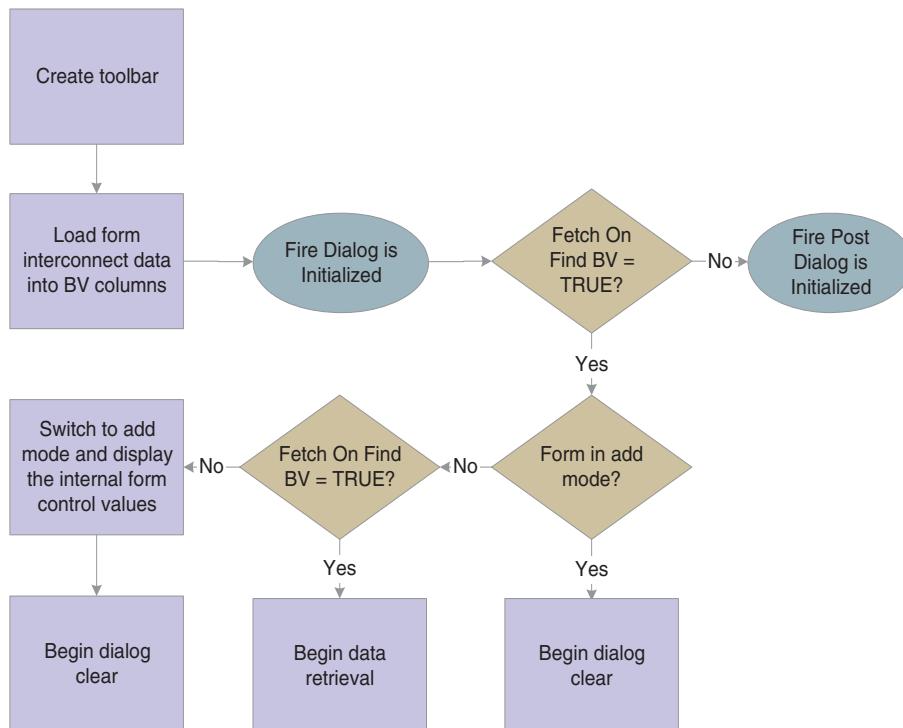
When a fix/inspect form is called, runtime initializes these items in this order:

1. Thread handling

2. Error handling process
3. Business view columns (BC)
4. Form controls (FC)
5. Static text
6. Helps
7. Event rules structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 6–1 Fix/Inspect dialog initialization



Note: FC and BC are sharing internal memory, so copying into the BV is effectively copying into the internal FC memory locations.

6.3.2 Dialog Clear

This list discusses how runtime clears the form in preparation to display retrieved data:

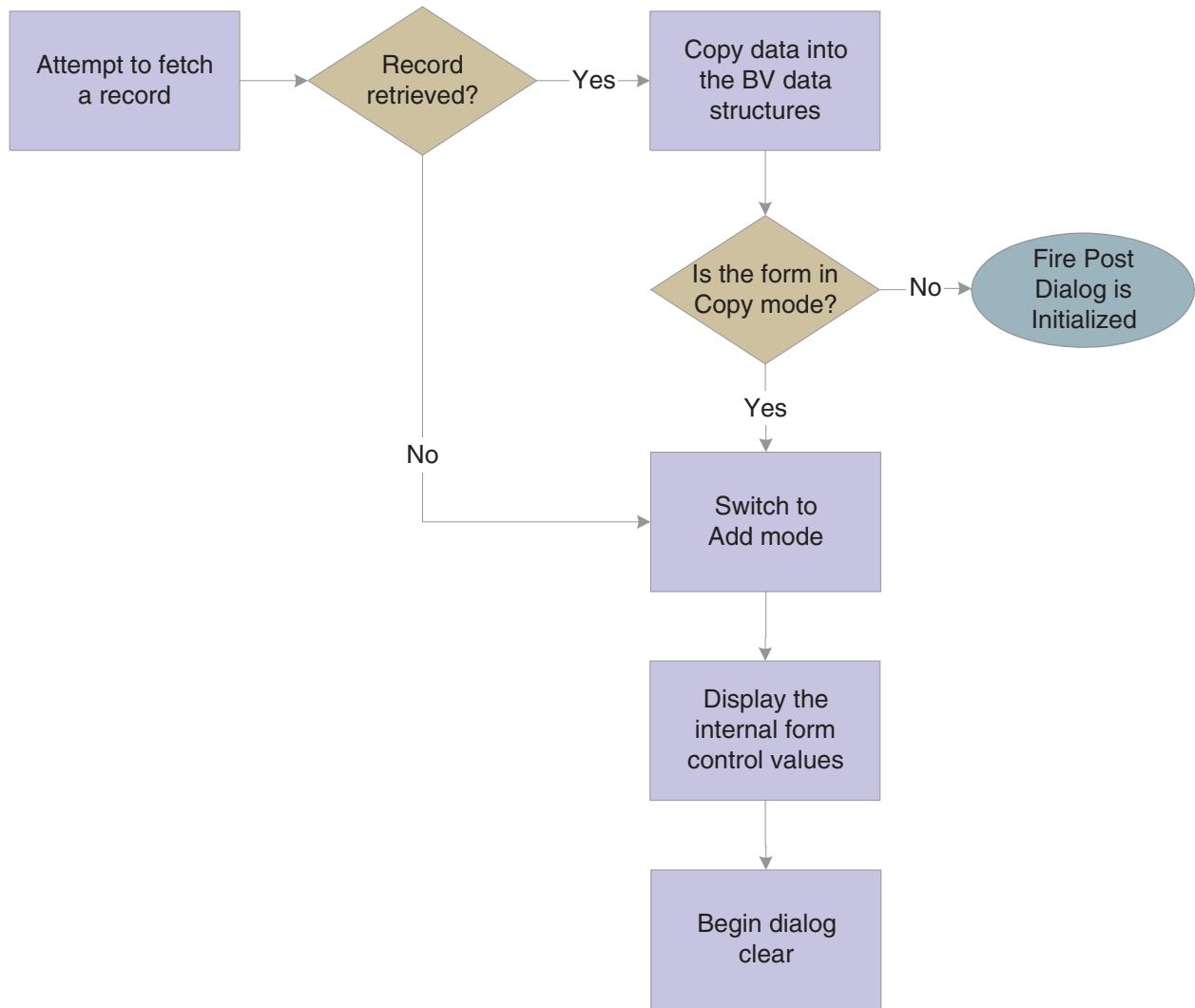
1. If the form was called in Copy mode, clear the key (primary index) controls for which the Do not clear after add option was enabled.
2. If the form was not called in Copy mode, clear all FCs for which the Do not clear after add option has been disabled.
3. Fire the **Clear Screen Before Add** event.
4. Fire the **Post Dialog is Initialized** event.

6.3.3 Data Retrieval

A request is issued to the JDEKRNL, which performs the actual fetch from the database. The fetch is based on the information passed to the form through its form data structure.

This flowchart illustrates the tasks that runtime performs to retrieve and display data on the fix/inspect form:

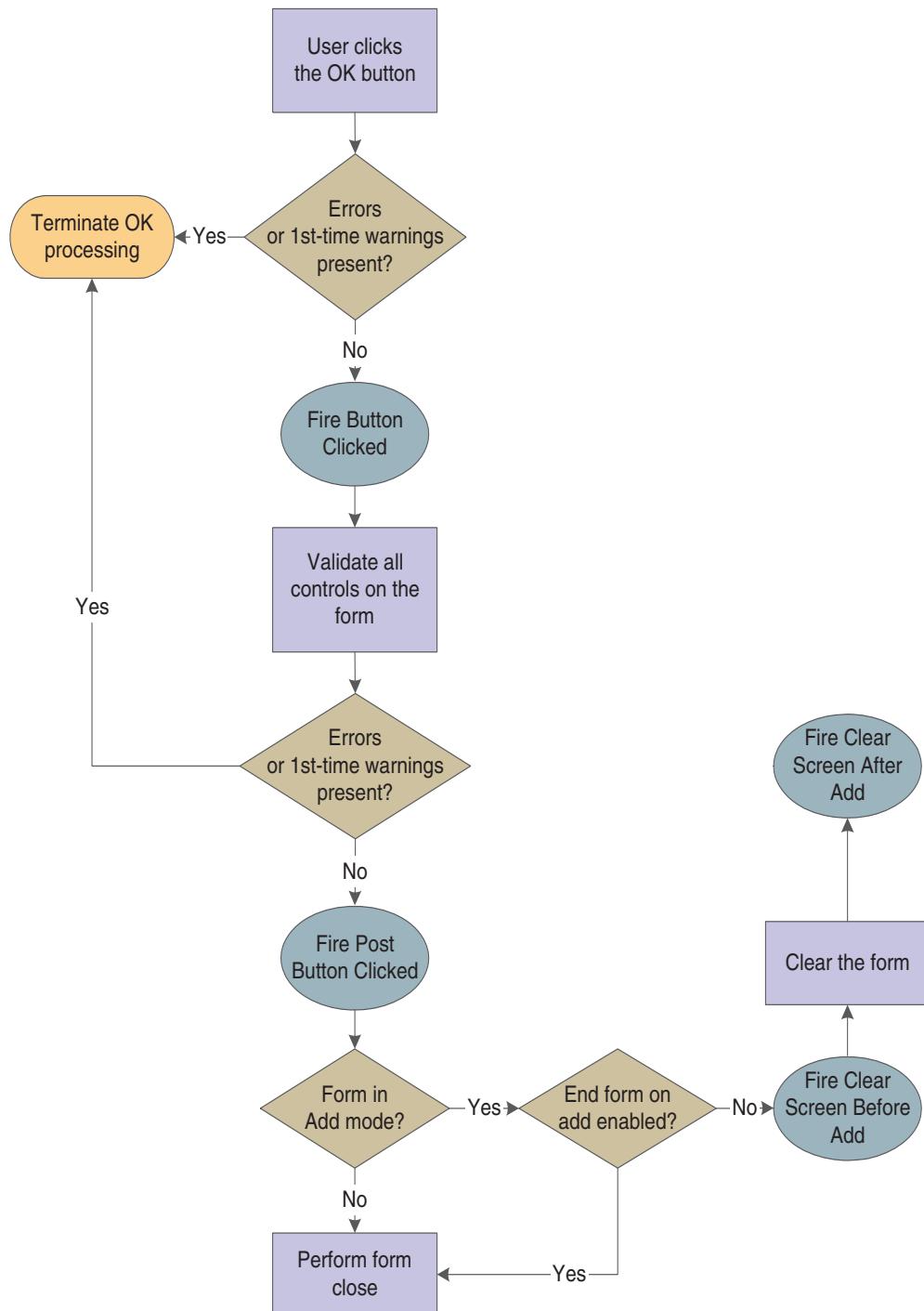
Figure 6–2 Fix/Inspect data retrieval



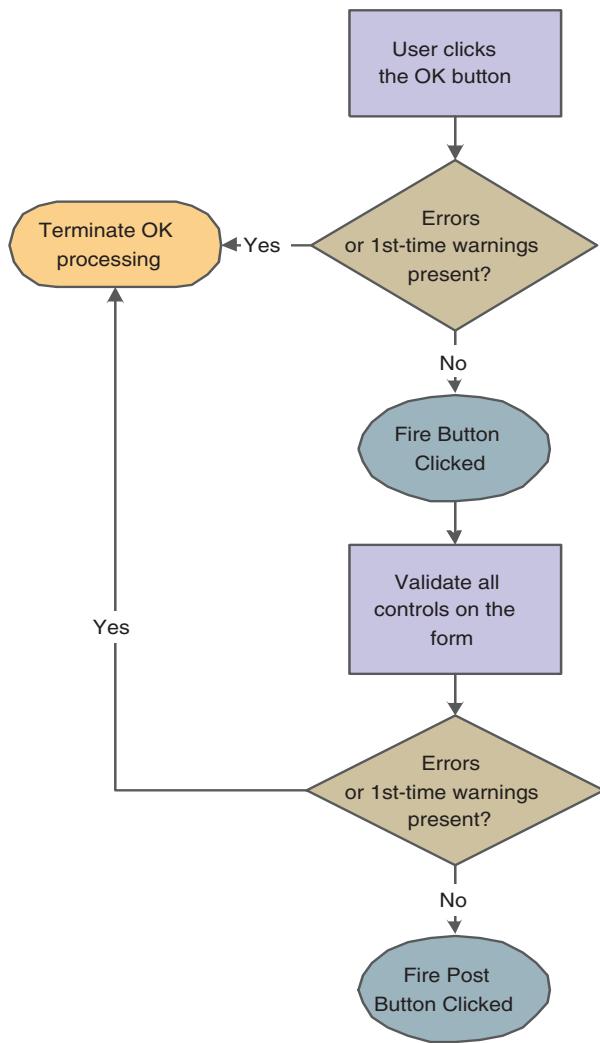
6.3.4 OK Button

The OK button is a standard button on fix/inspect forms that appears by default. It causes runtime to validate the information on the form and update or add it to the database through JDEKRNL function calls.

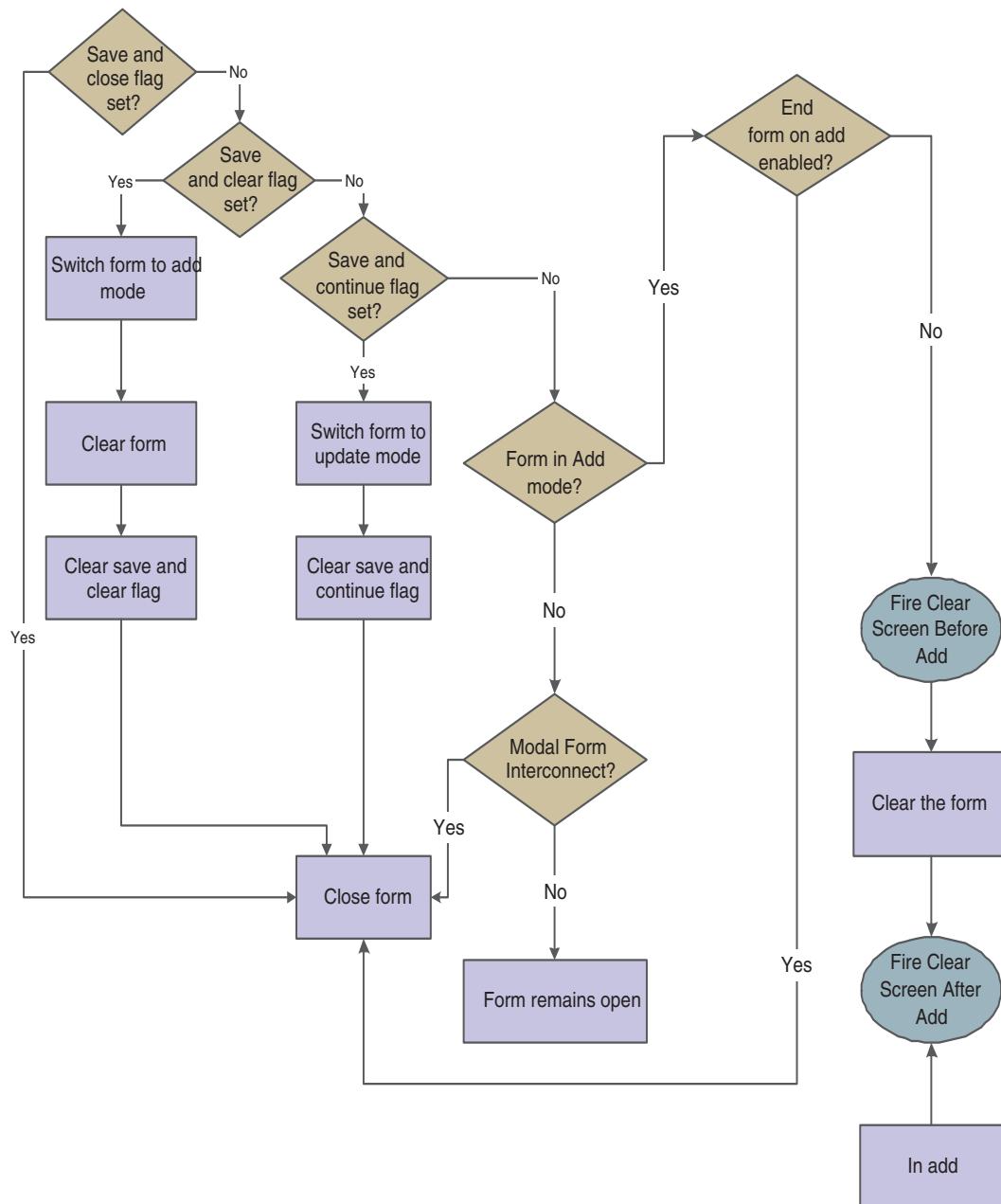
This flowchart illustrates the initial tasks that constitute runtime processing for the OK button:

Figure 6–3 Fix/inspect OK button processing, part 1 of 4

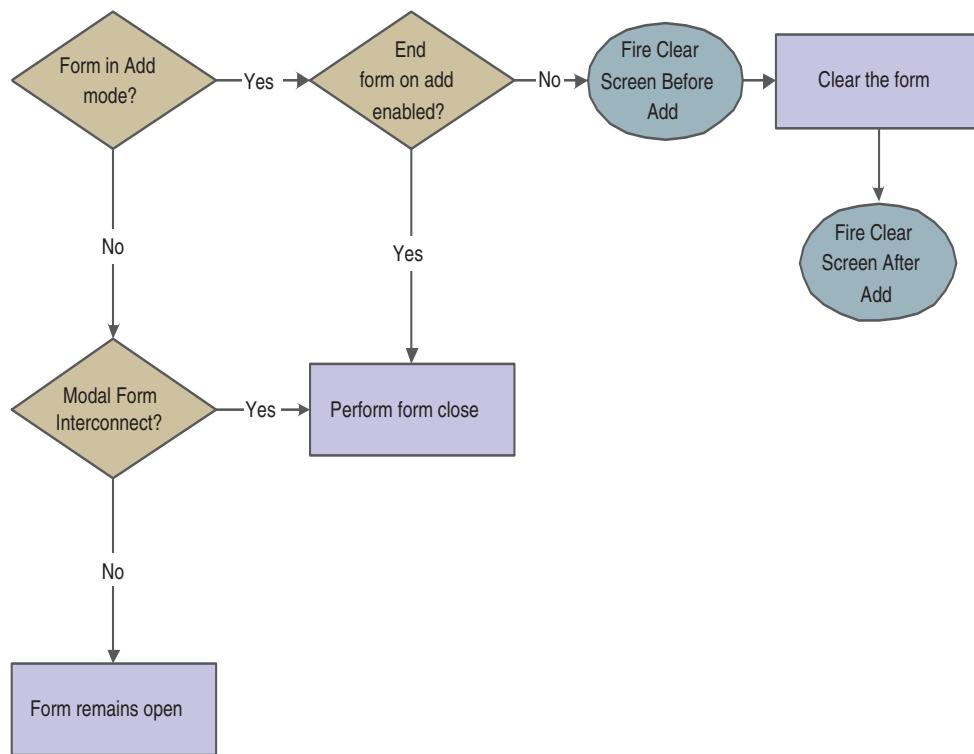
This flowchart expands on how runtime populates the grid during OK button processing:

Figure 6–4 Fix/inspect OK button processing, part 2 of 4

This flowchart illustrates how runtime processes any mode changes during OK button processing:

Figure 6–5 Fix/inspect OK button processing, part 3 of 4

This flowchart illustrates how runtime completes the form close process during OK button processing:

Figure 6–6 Fix/inspect OK button processing, part 4 of 4

6.3.5 Cancel Button

The Cancel button is a standard button on fix/inspect forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime attempts to close all of the modeless child forms, if any exist. If any of these child forms cannot be closed, the Cancel button process is terminated. Otherwise, runtime fires the **End Dialog** event and initiates the dialog close process.

6.3.6 Dialog Close

Fix/Inspect can be closed either by the user (typically by clicking the OK or Cancel buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BC for database commit.
2. Terminate error and thread handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

Understanding Header Detail Forms

This chapter contains the following topics:

- [Section 7.1, "Header Detail Forms"](#)
- [Section 7.2, "Header Detail Design-Time Considerations"](#)
- [Section 7.3, "Header Detail Events"](#)
- [Section 7.4, "Header Detail Runtime Processing"](#)

7.1 Header Detail Forms

A header detail form is used when a relationship exists between the information in the *header* (the fields above the grid) and the information in the *detail area* (the grid itself). The header portion uses one business view (BV), and the detail portion of the form can use another BV. This form type (as well as the headerless detail form type) is referred to as a *transaction form*.

7.2 Header Detail Design-Time Considerations

These property values are particularly significant in the design of the header detail form:

- Fetch on Form Businessview
- Fetch on Grid Businessview
- Update on Form Businessview
- Update on Grid Businessview

The fetch properties are important because of their effect in data display during runtime. You must select Fetch on Form Businessview to populate the controls in the header portion of the form. Likewise, you must select Fetch on Grid Businessview to populate the grid.

Similarly, select Update on Form Businessview if you intend to permit users to commit their modifications to records in the header controls to the database. Select Update on Grid Businessview to enable the commitment of modified grid records.

7.3 Header Detail Events

These events can occur on the header detail form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized

- Grid Record is Fetched
- Last Grid Record Has Been Read
- Clear Screen Before Add
- Clear Screen After Add
- Write Grid Line-Before
- Write Grid Line-After
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Post Commit
- End Dialog
- XAPI Subscribe Event

7.4 Header Detail Runtime Processing

This section discusses how runtime processes header detail forms. This section discusses form-level runtime processing only. Much of the runtime processing for the header detail "form" actually occurs on the level of the grid control, however.

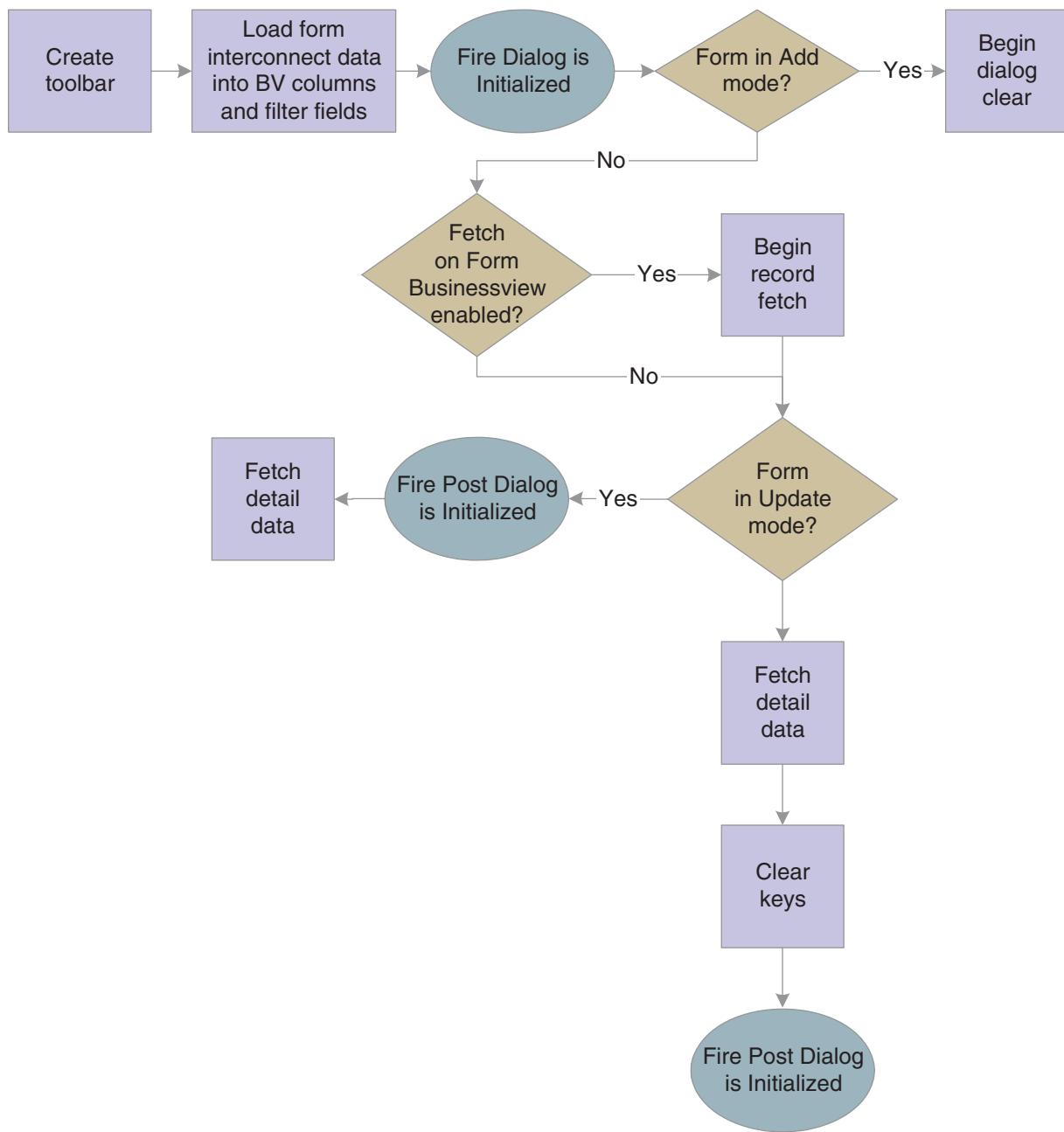
See [Grid Control Runtime Processing](#).

7.4.1 Dialog Initialization

When a header detail form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. BV columns
4. Form controls
5. Grid fields
6. Static text
7. Helps
8. Event rules (ER) structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 7–1 Header detail form initialization

7.4.2 Dialog Clear

This list discusses how runtime clears the form in preparation to display retrieved data:

1. If the form was called in Copy mode, clear the key (primary index) controls for which the Do not clear after add option was enabled.
2. If the form was not called in Copy mode, clear all form controls for which the Do not clear after add option has been disabled.
3. Fire the **Clear Screen Before Add** event.
4. Fire the **Post Dialog is Initialized** event.

7.4.3 Data Retrieval

After dialog initialization, runtime populates the form only if it is in Update mode. Then, it seeks to populate the header if the Fetch on Form Businessview option is enabled. If enabled, runtime fetches the header record and populates the controls in the header.

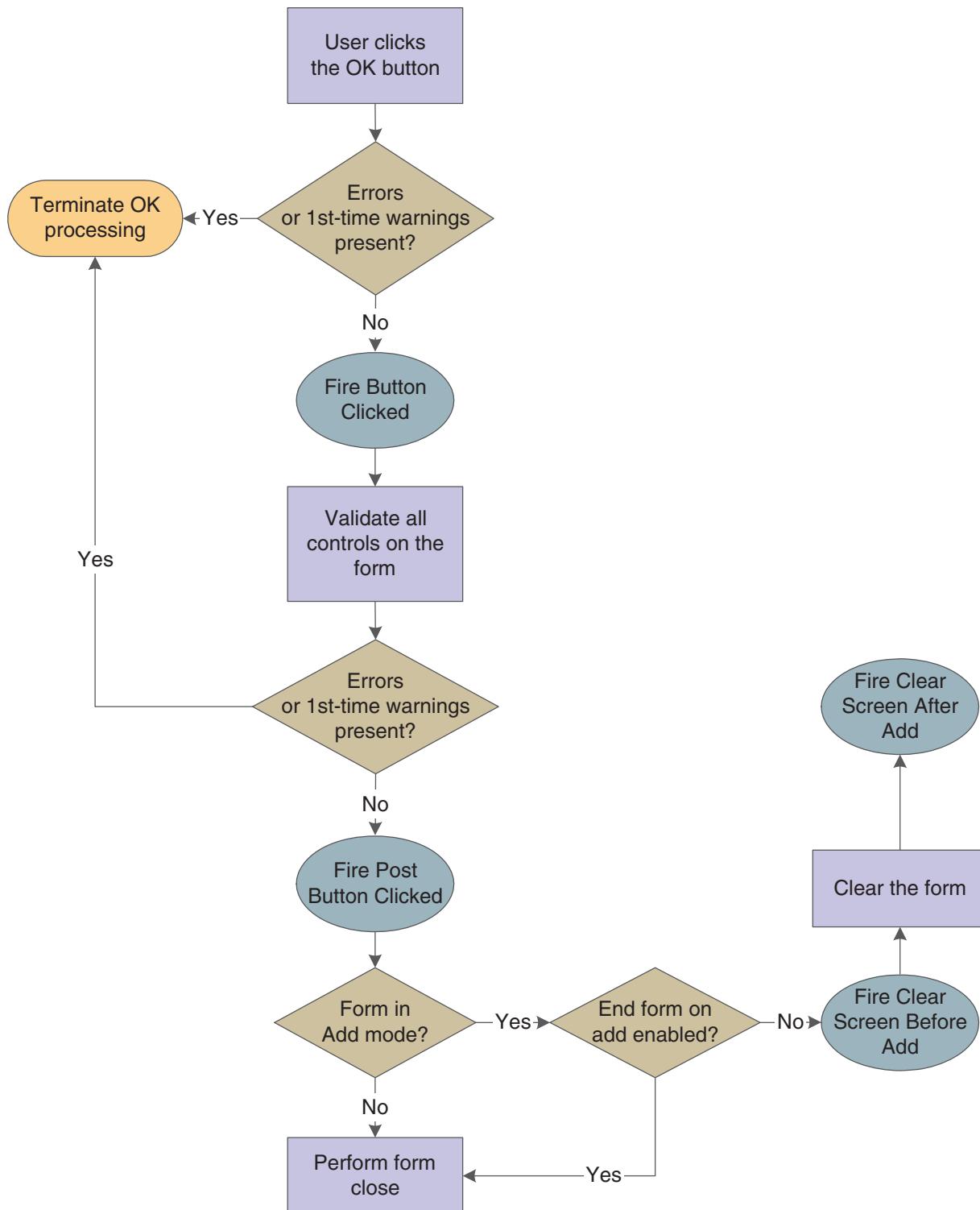
Next, runtime performs a fetch and populates the grid if the Automatically Find on Entry and the Fetch on Grid Businessview options are both enabled. The fetch and populate process used is standard for grid controls.

See [How Runtime Processes the Grid Control](#).

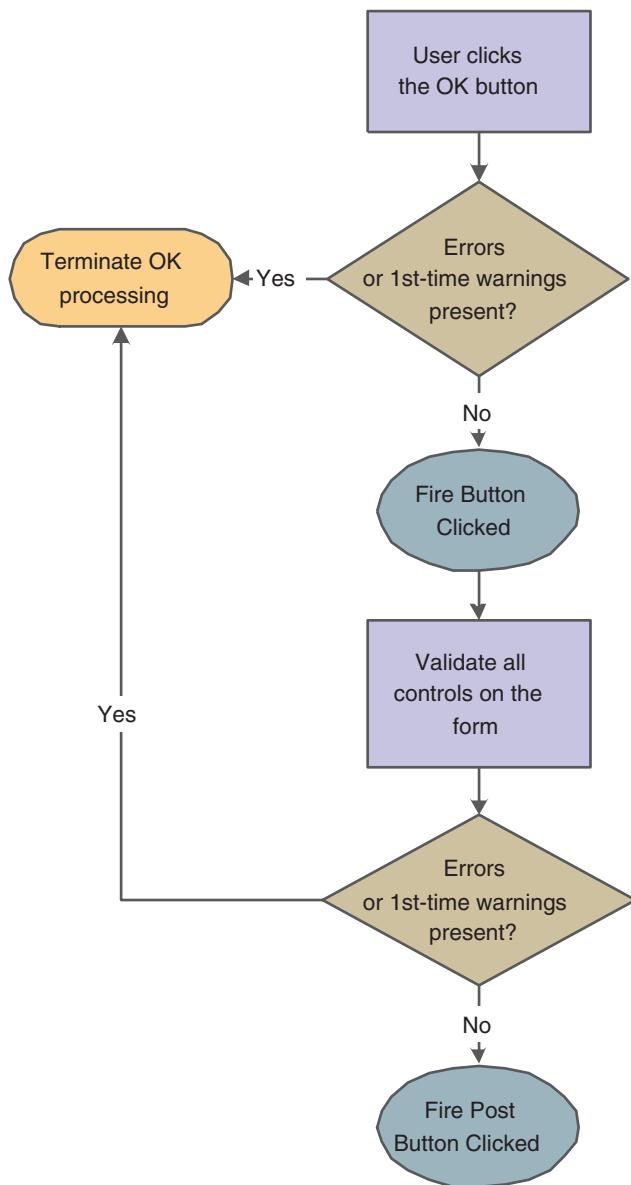
7.4.4 OK Button

OK is a standard button on header detail forms that appears by default. It causes runtime to validate the information on the form and update or add it to the database through JDEKRNL function calls.

This flowchart illustrates the initial tasks that constitute runtime processing for the OK button:

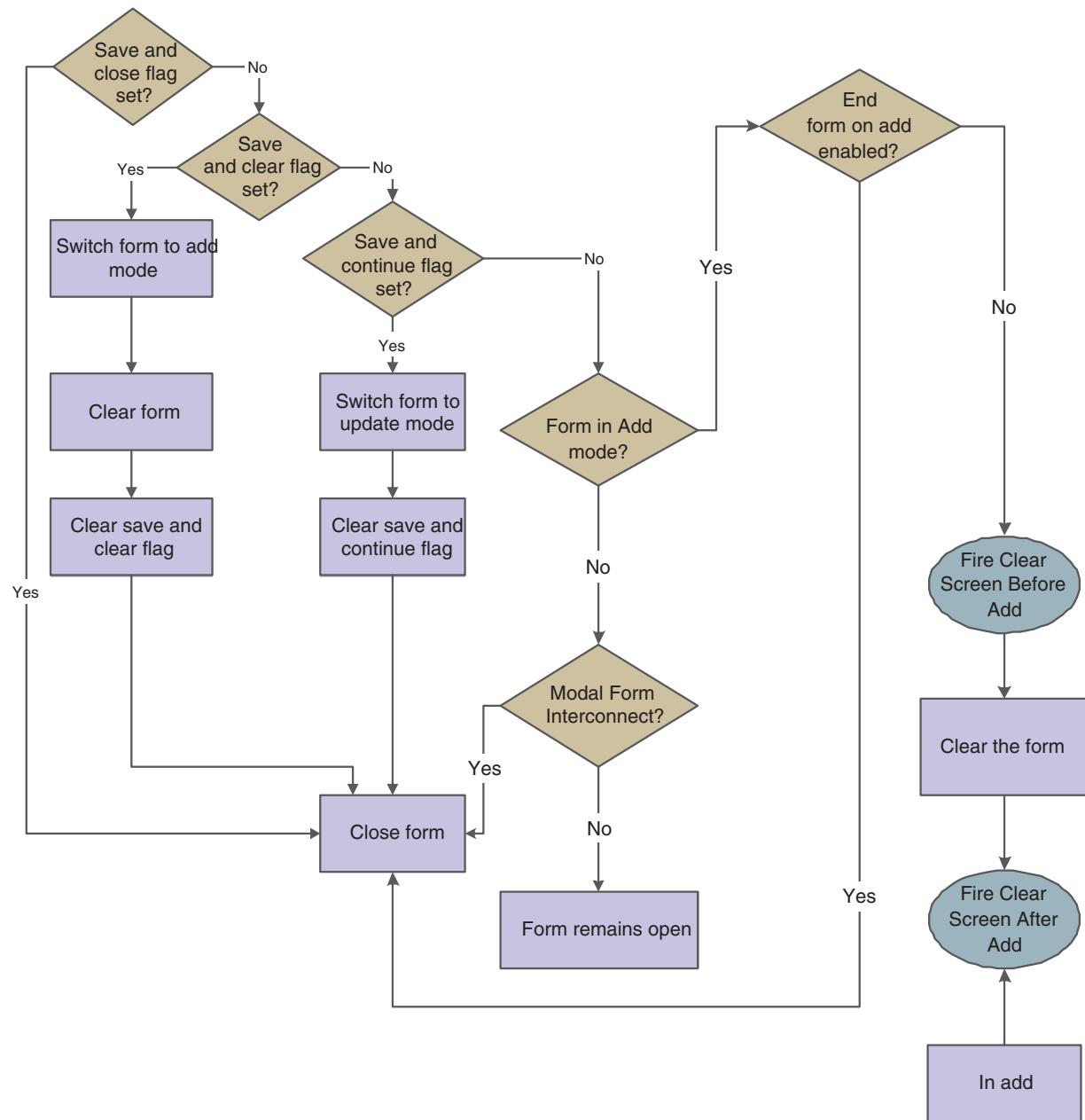
Figure 7–2 Header detail OK button processing, part 1 of 4

This flowchart expands on how runtime populates the grid during OK button processing:

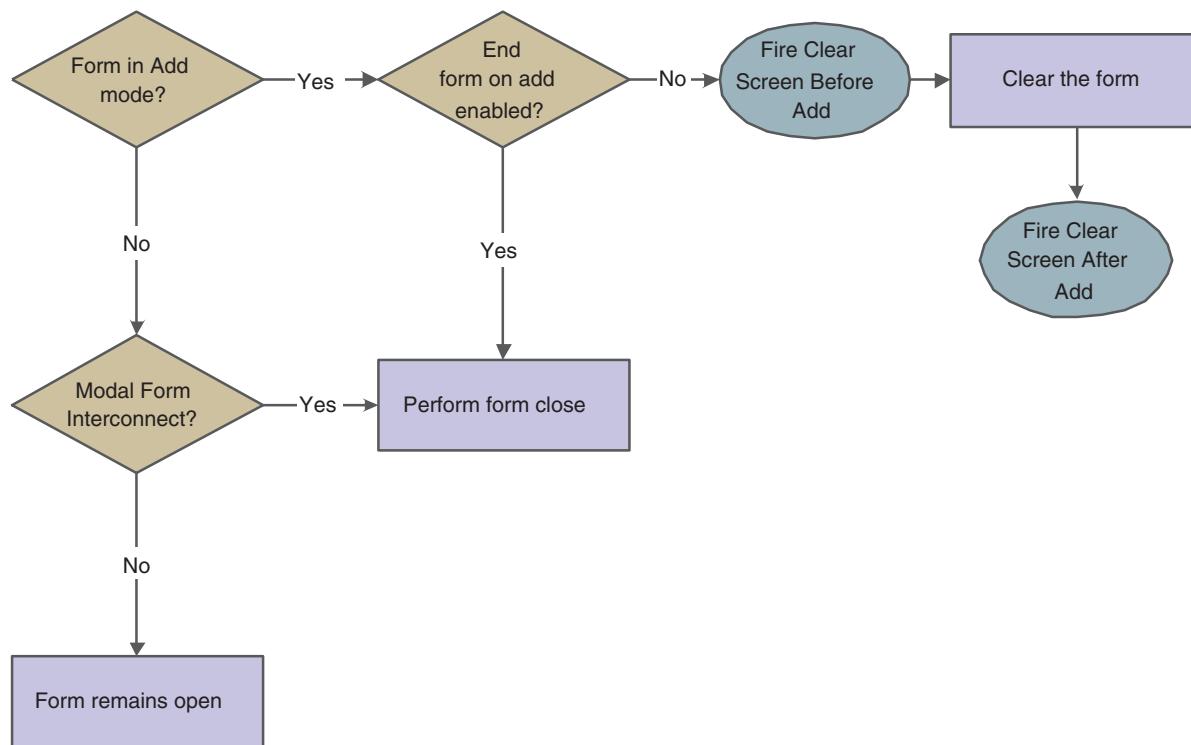
Figure 7–3 Header detail OK button processing, part 2 of 4

This flowchart illustrates how runtime processes any mode changes during OK button processing:

Figure 7-4 Header detail OK button processing, part 3 of 4



This flowchart illustrates how runtime completes the form close process during OK button processing:

Figure 7–5 Header detail OK button processing, part 4 of 4

7.4.5 Cancel Button

The Cancel button is a standard button on header detail forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime cancels any media objects that might be open. Also, if a manual transaction is in process, runtime attempts to cancel it as well. Then, runtime fires the **End Dialog** event and initiates the dialog close process.

7.4.6 Dialog Close

Header detail can be closed either by the user (typically by clicking the OK or Cancel buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BV columns for database commit.
2. Terminate error and thread handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

Understanding Headerless Detail Forms

This chapter contains the following topics:

- [Section 8.1, "Headerless Detail Forms"](#)
- [Section 8.2, "Headerless Detail Design-Time Considerations"](#)
- [Section 8.3, "Headerless Detail Events"](#)
- [Section 8.4, "Headerless Detail Runtime Processing"](#)

8.1 Headerless Detail Forms

A headerless detail form is used to update and enter records that have information that is common to all records in a selected group. This form type (as well as the header detail form type) is referred to as a *transaction form*.

8.2 Headerless Detail Design-Time Considerations

These property values are particularly significant in the design of the headerless detail form:

- Fetch on Grid Businessview
- Update on Grid Businessview

The fetch property is important because of its effect in data display during runtime. You must enable Fetch on Grid Businessview to populate the grid. Similarly, enable Update on Grid Businessview to enable the commitment of modified grid records.

8.3 Headerless Detail Events

These events can occur on the headerless detail form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Last Grid Record Has Been Read
- Clear Screen Before Add
- Clear Screen After Add
- Write Grid Line-Before
- Write Grid Line-After

- Post Commit
- End Dialog
- XAPI Subscribe Event

8.4 Headerless Detail Runtime Processing

This section discusses how runtime processes headerless detail forms. This section discusses form-level runtime processing only. Much of the runtime processing for the headerless detail "form" actually occurs on the level of the grid control, however.

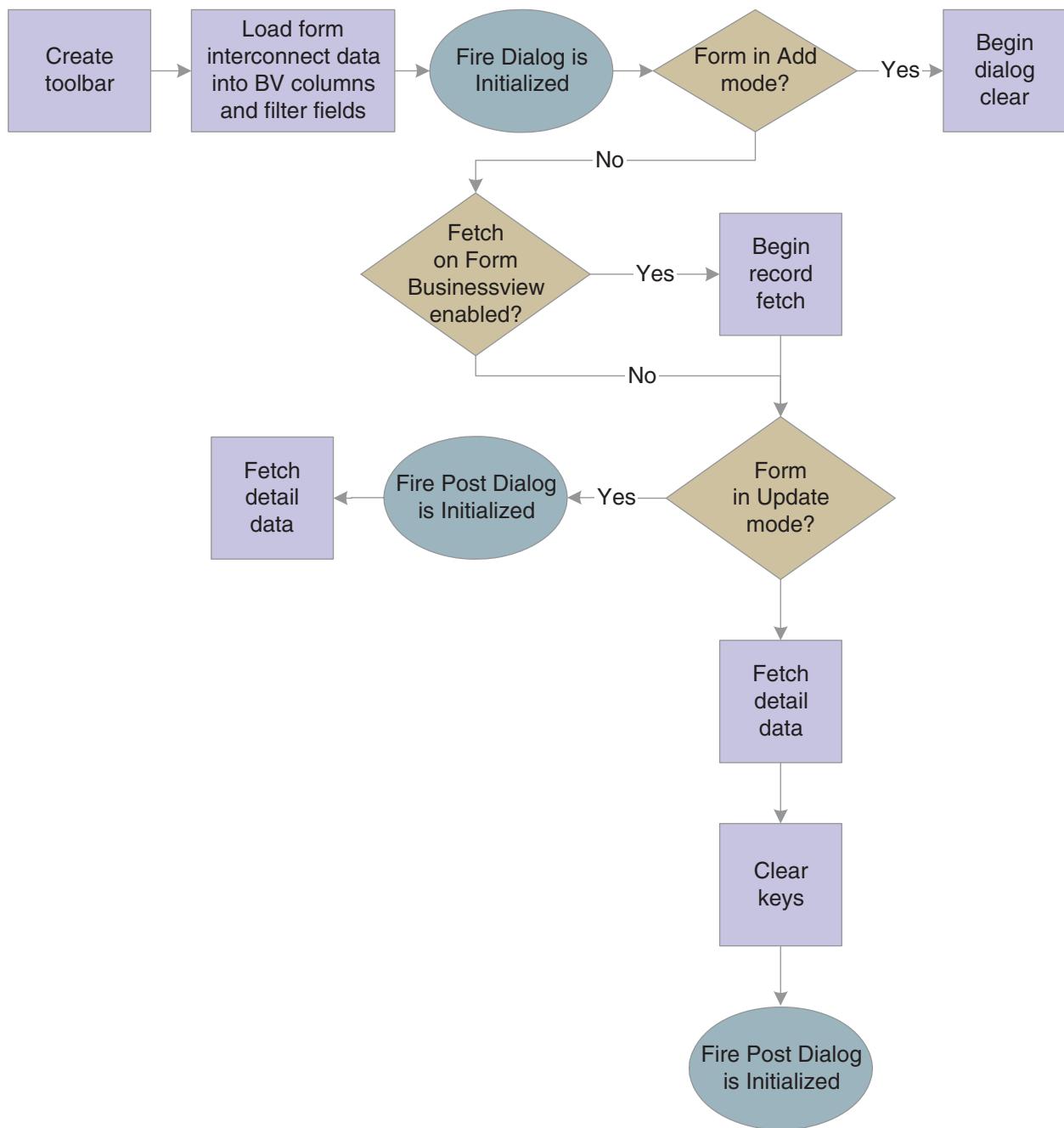
See [Grid Control Runtime Processing](#).

8.4.1 Dialog Initialization

When a headerless detail form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. Business view columns (BCs)
4. Form controls (FCs)
5. Grid fields
6. Static text
7. Helps
8. Event rules (ER) structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 8-1 Headerless detail dialog initialization

8.4.2 Dialog Clear

This list discusses how runtime clears the form in preparation to display retrieved data:

1. If the form was called in Copy mode, clear the key (primary index) controls for which the Do not clear after add option was enabled.
2. If the form was not called in Copy mode, clear all FCs for which the Do not clear after add option has been disabled.
3. Fire the **Clear Screen Before Add** event.

4. Fire the **Post Dialog is Initialized** event.

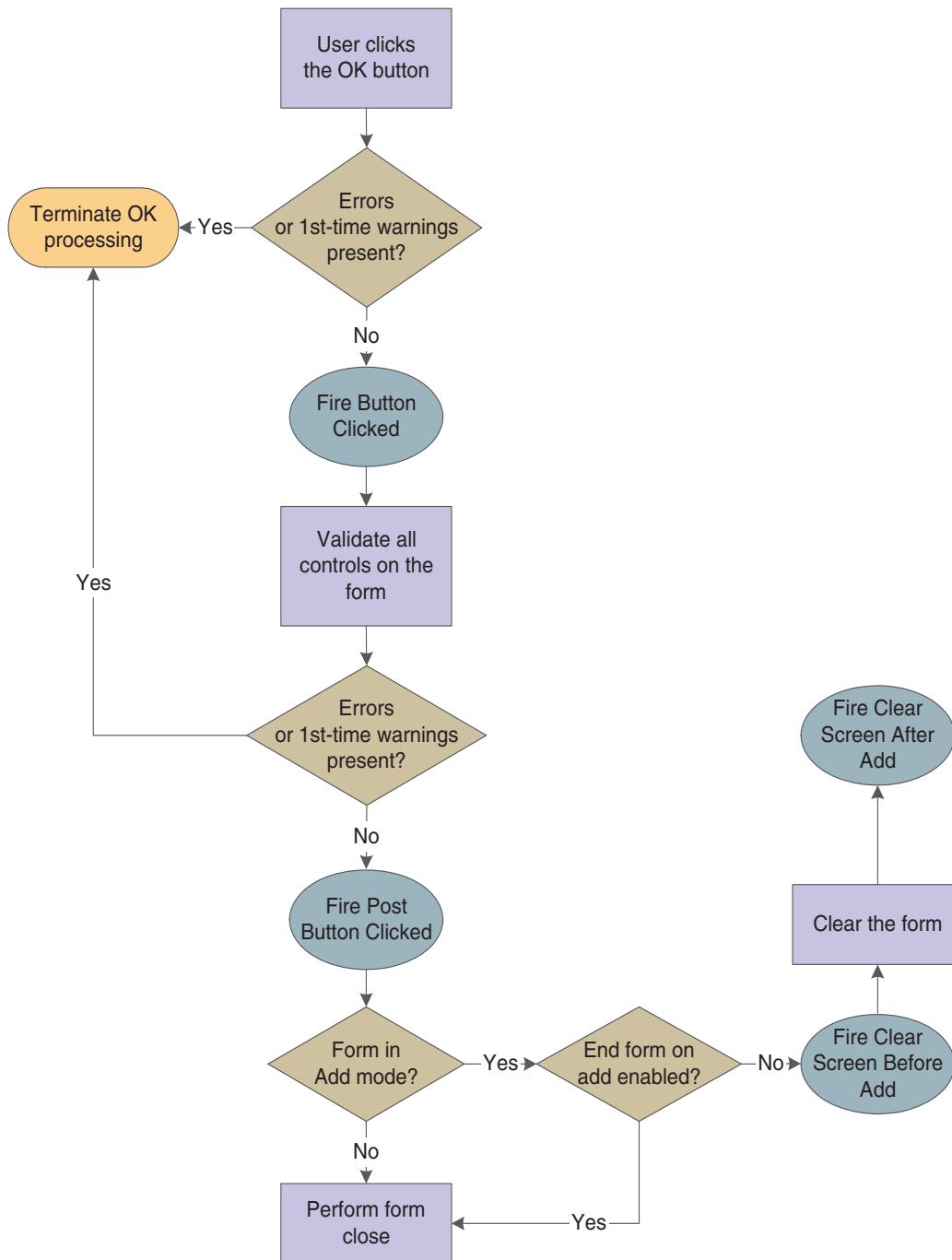
8.4.3 Find Button

The Find button is a standard button on headerless detail forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the filter fields, runtime performs data selection and sequencing for the grid control. After reloading the grid with the fetched data, runtime fires the **Post Button Clicked** event.

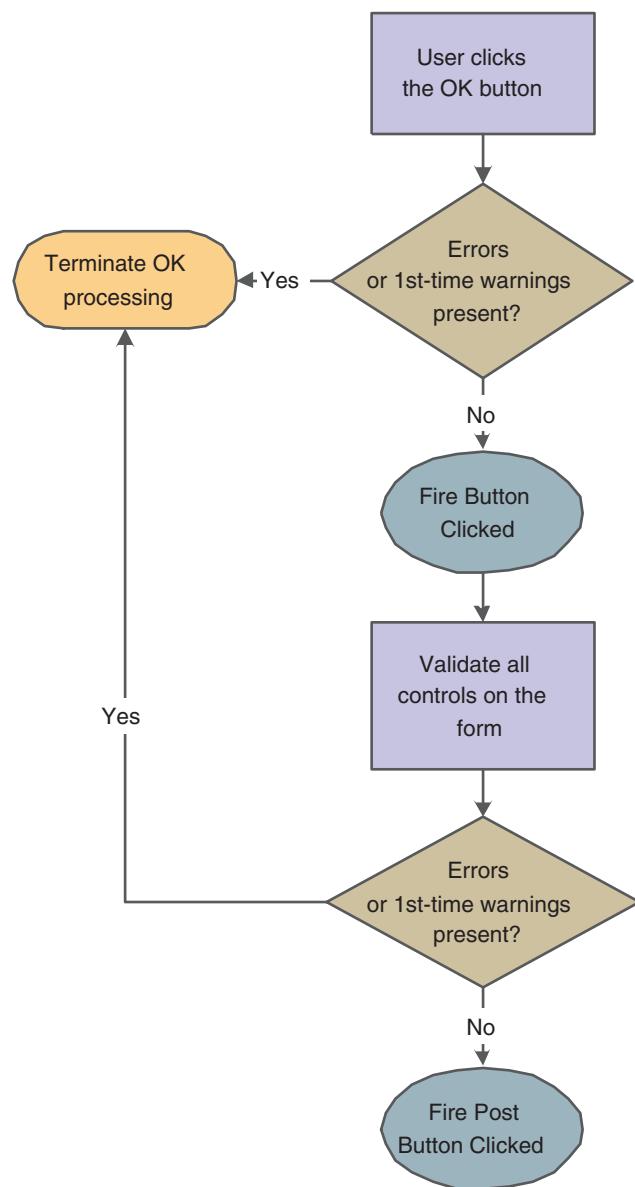
8.4.4 OK Button

OK is a standard button on headerless detail forms that appears by default. It causes runtime to validate the information on the form and update or add it to the database through JDEKRNL function calls.

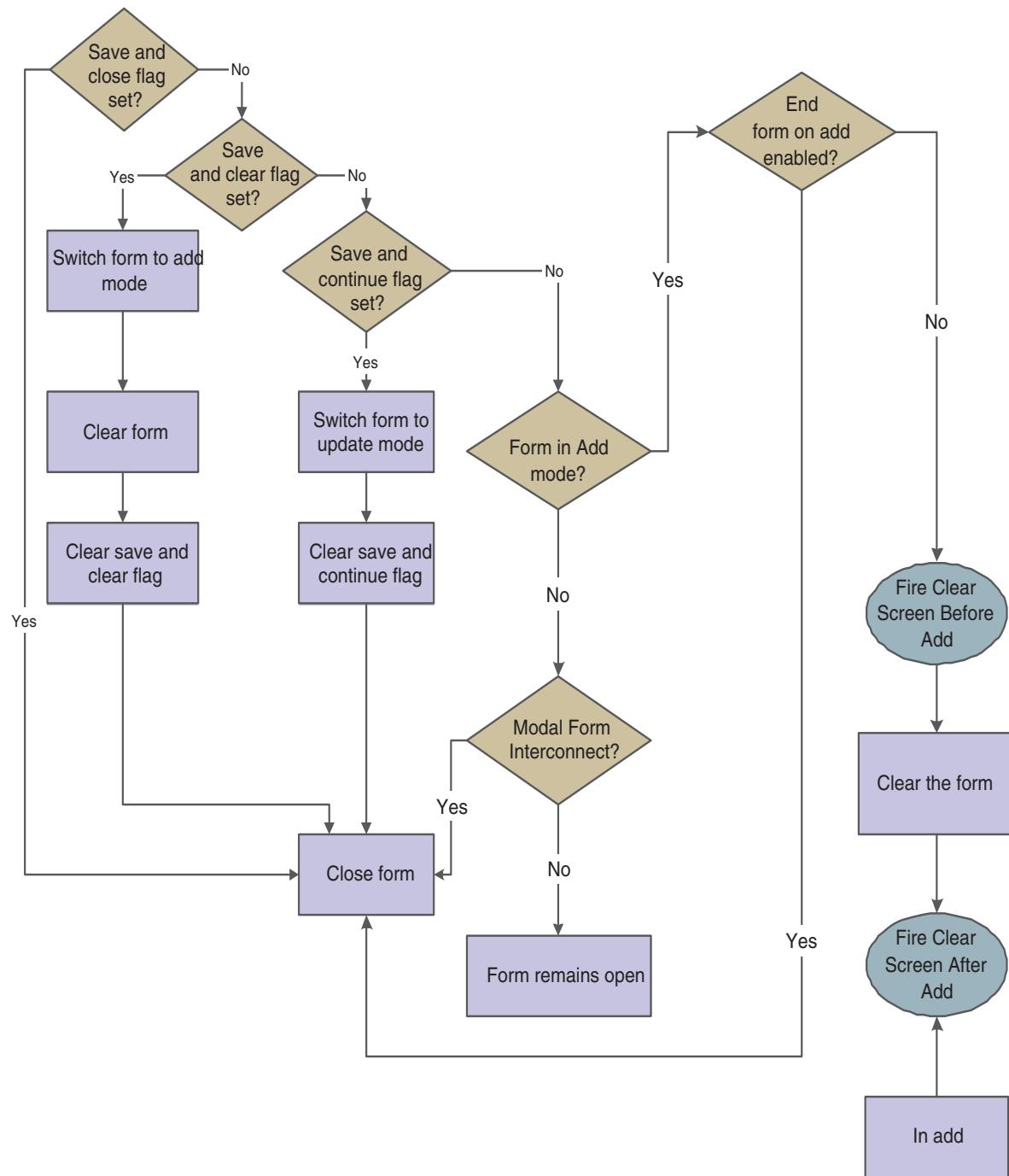
This flowchart illustrates the initial tasks that constitute runtime processing for the OK button:

Figure 8–2 Headerless detail OK button processing, part 1 of 4

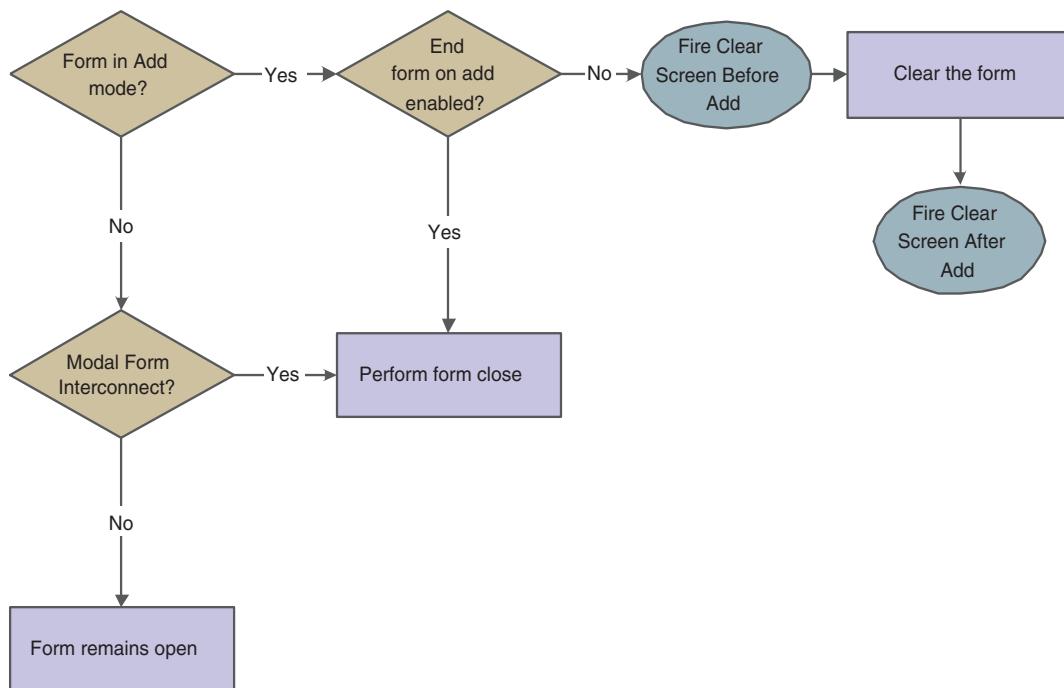
This flowchart expands on how runtime populates the grid during OK button processing:

Figure 8–3 Headerless detail OK button processing, part 2 of 4

This flowchart illustrates how runtime processes any mode changes during OK button processing:

Figure 8–4 Headerless detail OK button processing, part 3 of 4

This flowchart illustrates how runtime completes the form close process during OK button processing:

Figure 8–5 Headerless detail OK button processing, part 4 of 4

8.4.5 Delete Button

The Delete button is a standard button on headerless detail forms that appears by default. The actual delete from the database does not occur at this point. Runtime verifies the intention to delete when the user clicks the Delete button, and then commits the deletion when the user clicks the OK button. Consequently, if the user clicks the Cancel button, the records are not purged from the database.

8.4.6 Cancel Button

The Cancel button is a standard button on headerless detail forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime cancels any media objects that might be open. Also, if a manual transaction is in process, runtime attempts to cancel it as well. Then, runtime fires the **End Dialog** event and initiates the dialog close process.

8.4.7 Dialog Close

Headerless detail can be closed either by the user (typically by clicking the OK or Cancel buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BC for database commit.
2. Terminate error and thread handling.
3. Terminate helps.
4. Free all form structures.

5. Destroy the window.

Understanding Message Forms

This chapter contains the following topics:

- [Section 9.1, "Message Forms"](#)
- [Section 9.2, "Message Form Design-Time Considerations"](#)
- [Section 9.3, "Understanding Message Form Events"](#)
- [Section 9.4, "Message Form Runtime Processing"](#)

9.1 Message Forms

The message form type is a form that appears as a secondary window to inform the user of something or to ask a question. It parallels the behavior of a Windows message box. The form does not have a tool bar or a status bar and can only contain static text and buttons.

Message forms permit only limited use of processing option (PO) values and business functions. Therefore, do not use this form type for complex logic.

You can use the message form as a hover form by selecting the pop-up option under the message form properties.

When the "Pop-up Form" check box is selected, all the action buttons, such as OK and Cancel, are hidden. A message form marked as a pop-up form acts as a plain container for the controls. The following controls are supported on a message form that is marked as a pop-up form: Label, Text Block, Group Box, and Image.

When the user hovers on the orange dot and clicks the hover indicator in the hover supported control, a pop-up window with context-based information is displayed.

See "Message Form as Hover Form" in *JD Edwards EnterpriseOne Tools Foundation Guide*.

The hover form can be associated with the selected controls using the system function Show Popup and Mouse is Hovered event or Feature Authorization applications.

See "Understanding Hover Forms" in the *JD Edwards EnterpriseOne Tools Runtime Administration Guide*.

The application developer can associate the hover form to the hover form supported form controls, grid row, and grid cell by calling the Show Popup system function in the Mouse is Hovered event.

9.1.1 Hover Events

Mouse is Hovered

This event is provided for hover form supported form controls; grid, and grid columns. This event is used to associate a hover form using Show Popup system function on form controls, grid rows or grid cells.

9.1.2 Hover System Functions

Show Popup (Web Only) System Function

This system function is provided to show the hover form when user hovers on a form control, a grid row selector or grid cell. This system function is listed in the General section of the Event Rules system function listing.

Parameters

<Popup Form>

There are two options to choose a pop up form:

1. Select Pop Up Form: Use this option to select a specific pop-up form. Once you click on this option, the system will let you choose the Application Name, Form Name, Version, and the parameters to pass into the selected hover pop up form.
2. Default Popup: Use this option to select the default pop-up option for grid headers. When using the Show Popup (Web Only) system function on the Mouse is Hovered event on the Grid control itself and you choose the Default Popup option, the system will display the grid row data in a tabular format on the hover pop up form. When user hovers on a grid row header, the system will display that particular row data in a tabular format within the hover form. This way, you can quickly glance at a row's values without scrolling the row if it has many columns.

Get Hovered Row Number

This system function returns the current hovered row number when you hover on a hover form associated grid cell.

Parameters

- Grid :
Input, required. The grid control on which the hover form is associated on a particular cell.
- Row
Output, required. Currently hovered row number - return value from the system function.

GetCollaborate

This system function is provided to create the HTML content for the Collaborate tab with the List of contact names and defined email addresses for the address book numbers.

Parameters

- TEXTBLOCKCONTROL
Input, required. FC text block control to effect.
- SEGMENT ID
Input, required. Segment of the text block control to effect.
- Address Number

- Input, required. The address number for which the contacts have to be fetched from Who's Who table.
- Collaborate Content
Output field. This string field contains the HTML content for the Collaborate tab with the List of contact names and defined email addresses for the address book numbers.
 - Show Parameterized URL
Input, required. Flag to indicate whether to show or hide the parameterized URL in the email or calendar request.

Note: With Release 9.2.1, for existing hover forms that call the Get Collaborate system function, you can create a "Collaborate" tab and drop a subform that contains a grid on the tab. In the Grid Properties, under the HTML Options tab, enter "5" for the Grid Row Count. The form should be over a business view that is a simple join between F0111 and F01151 by column AN8, distinct IDLN, with a filter F01151.ETP equals "E".

With Release 9.2.1, use the Hover Form property on the Power Browse form, to create new hover forms.

See [Section 12.3, "Power Browse Form Design-Time Considerations"](#).

9.2 Message Form Design-Time Considerations

Message forms are unique among JD Edwards EnterpriseOne forms in that they include a default push button control on the form. You can configure this button to be OK, Cancel, Yes, or No. You can also make it of type Other and equip it with special functions of your own. Because they have no tool bar, message forms do not have the standard menu buttons available to the other forms. Hence, you must provide all functions with manually-added controls. Because message forms are modal windows, you cannot access the controls on the calling form.

The default size for a message form is 64 pixels high and 273 pixels wide. To accommodate different screen sizes or resolutions, you can change it in the Properties form.

9.3 Understanding Message Form Events

These events can occur on the message form during runtime:

- Dialog is Initialized
- XAPI Subscribe Event

9.4 Message Form Runtime Processing

This section discusses how the runtime engine processes the message form.

9.4.1 Dialog Initialization

When a message form is called, runtime initializes these items in this order:

1. Form controls

2. Static text
3. Helps
4. Event rules (ER) structures

Then, it fires **Dialog is Initialized**.

9.4.2 Dialog Close

Message forms close when the user clicks the default button. Then runtime performs these tasks in this order:

1. Terminate helps.
2. Free all form structures.
3. Destroy the window.

10

Understanding Parent/Child Browse Forms

This chapter contains the following topics:

- [Section 10.1, "Parent/Child Browse Forms"](#)
- [Section 10.2, "Parent/Child Browse Events"](#)
- [Section 10.3, "Parent/Child Browse Runtime Processing"](#)

10.1 Parent/Child Browse Forms

Similar to find/browse forms, parent/child browse forms are used to query business views (BVs) and select records from BVs for operations. However, instead of a default grid control, parent/child browse forms contain a default parent child control instead.

10.2 Parent/Child Browse Events

These events can occur on the parent/child browse form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Write Grid Line-Before
- Write Grid Line-After
- Last Grid Record Has Been Read
- XAPI Subscribe Event
- End Dialog

10.3 Parent/Child Browse Runtime Processing

This section discusses how runtime processes parent/child browse forms.

10.3.1 Dialog Initialization

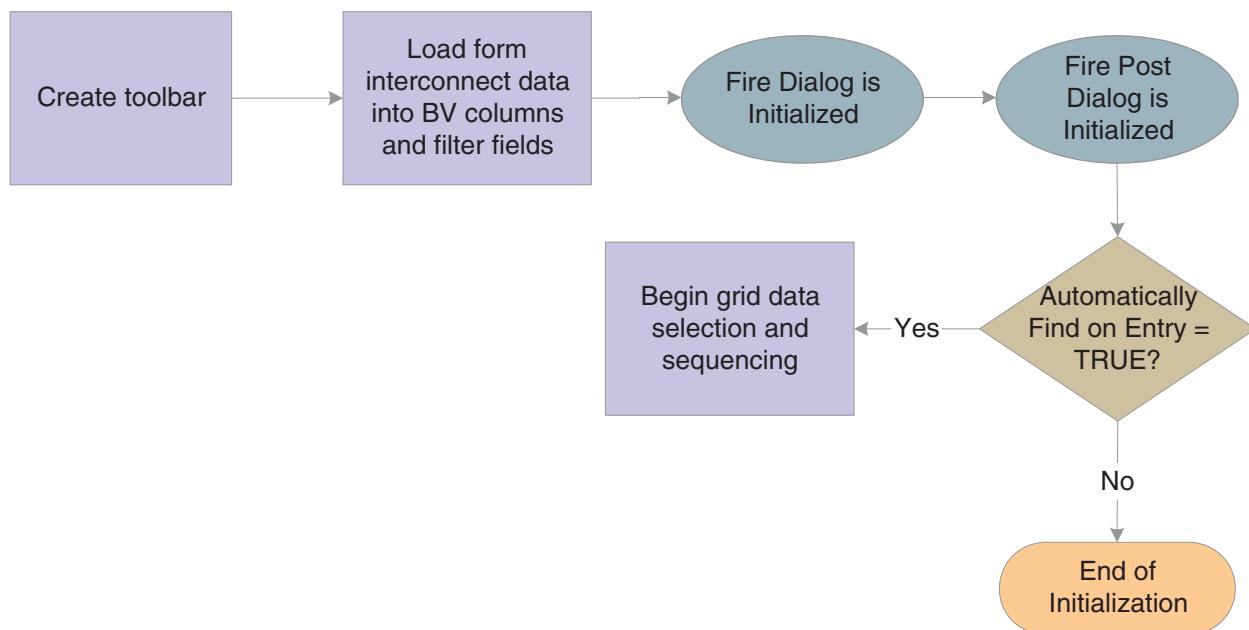
When a parent/child browse form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. Business view columns (BCs)

4. Form controls (FCs)
5. Grid fields
6. Static text
7. Helps
8. Event rules (ER) structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 10–1 Parent/Child browse dialog initialization



Data selection and sequencing occurs at the control level and ultimately leads to the population of the parent/child control, provided runtime encounters no errors.

10.3.2 Find Button

The Find button is a standard button on parent/child forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the filter fields, runtime performs data selection and sequencing for the grid and tree controls. After reloading the grid and tree with the fetched data, runtime fires the **Post Button Clicked** event.

10.3.3 Select Button

Select is a standard item that is automatically placed on parent/child browse forms. No default processing exists for Select on parent/child browse forms. It acts as a user-defined item.

10.3.4 Close Button

The Close button is a standard button on parent/child forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked**

events in immediate succession. If no errors occur, runtime attempts to close all of the modeless child forms, if any exist. If any of these child forms cannot be closed, the Close button process is terminated. Otherwise, runtime fires the **End Dialog** event and initiates the dialog close process.

10.3.5 Dialog Close

Parent/Child can be closed either by the user (typically by clicking the Close button) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BCs for database commit.
2. Terminate error and thread handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

Understanding Portlet Forms

This chapter contains the following topics:

- [Section 11.1, "Portlet Design Considerations"](#)
- [Section 11.2, "Portlet Types"](#)
- [Section 11.3, "Understanding Portlet Forms"](#)
- [Section 11.4, "Generating Portlets \(Prior to release 9.2.2\)"](#)
- [Section 11.5, "Updating an FDA-Created Portlet after Initial Installation \(Prior to release 9.2.2\)"](#)

11.1 Portlet Design Considerations

A *portal* is an application that aggregates portlets into pages and provides authentication, authorization and administration tools. A *portlet* is a single piece of portal content; it is the window through which users interact with the application while using a portal. Portlets provide information and act as gateways to full-blown applications. Portlets cannot communicate to one another, only to an application.

The principle value of a portlet to a user is that, compared to launching an application and searching for data manually, a portlet can significantly shorten the number of clicks required to get from the portal to the desired information or transaction.

11.2 Portlet Types

This section discusses the three portlet types:

- Alerts
- Menus
- Shortcuts

11.2.1 Portlets that are Alerts

These types of portlets are typical of "dashboard" portal configurations used for business intelligence. They show a single dimension of data, provide a description for that data, and have a threshold value, which is often user configurable. If the threshold is breached, some event usually occurs to alert the user to the condition.

As a rule of thumb, limit each portlet to a single data dimension. Users who want to monitor several dimensions should have several portlets available on their portal—one for each dimension to be monitored.

Often, these types of portlets also provide a link to a transactional application that enables the user to see more detail and then change the situation that has caused the alert. If you provide such a link, be sure to prepopulate the transactional application with the data driving the portlet display result. Ideally, the user should not need to perform any queries for the initial data view when the application launches.

11.2.2 Portlets that are Menus

These types of portlets contain task-based links and shortcuts to web locations and applications. The value of such portlets is that the links should be highly tailored to the job role. The links you create to applications should therefore preload as much data as possible, based on the user's role. The idea is to reduce the amount of repetitive data entry the user must perform before entering a frequently-used application.

For example, a sales representative might want a portlet with a menu comprising his or her top 10 accounts. By clicking an account, the user launches an application in a view that shows an overview of all account activity for the last three months.

11.2.3 Portlets that are Shortcuts

These types of portlets facilitate information lookup. Typically, they accept a limited amount of information, one or two fields of data. Just as with menu portlets, you should build in as much as possible based on the job role of the user to reduce data entry and target the result set as much as possible.

Frequently, these types of portlets display the results in an application, not the portlet itself. However, the application is often one that facilitates filtering the results and is not necessarily a full-blown transactional application. Based on your context, you might want to make it easy for the user to launch such an application from the filtering view.

11.3 Understanding Portlet Forms

This section discusses:

- Portlet form features.
- Portlet personalization.
- Portlet form events.
- Edit portlet form design-time considerations.

11.3.1 Portlet Form Features

Portlet forms are the form types you use to create portlets intended for use in the Oracle Portal or the Collaborative Portal.

Note: Portlet forms are supported in JD Edwards EnterpriseOne version 8.11 and might not function correctly in older versions of the software.

Portlet forms have these general characteristics:

- All regular controls can be placed on portlet forms.
- Multiple tab controls are permitted.

- Toolbar and form/row exits are not permitted on a portlet form.
- You must add action buttons to a portlet form.
No default action buttons are defined by Form Design Aid (FDA).
- Default actions can be placed anywhere on the portlet form.
- Portlet forms do not contain scroll bars; you must display all controls within the form.
- Portlet forms support the Fetch on Business View and Update on Business View form property options.
- You can nest reusable and embedded subforms in portlet forms.

JD Edwards EnterpriseOne Tools provides two types of portlet forms: *browse* portlets and *edit* portlets. Browse portlets enable the display of and queries for information. Edit portlets enable user edit functions—adding, changing, and deleting table information.

If you include a form interconnection in a portlet, the portal containing it will automatically resize the portlet to its maximum size when the user triggers the interconnection. When the user closes the application, the portlet returns to its normal size.

11.3.2 Portlet Personalization

One feature of portlet forms that is unique among FDA form types is portlet personalization. Users can toggle between standard mode (where they interact with the application) and personalization mode, where they can select different options that affect how the application runs while in standard mode.

For example, say that the application by default displays all accounts that are overdue by 30 days or more. Users might be able to choose instead to view accounts that are overdue by 60 days instead, or by accounts in a certain region that are overdue. Alternatively, they might have an option which displays the information in a summarized format instead of showing all the data you show by default.

Personalization values are user specific.

You control what options users have. Every option you provide, be it as simple as a filtering function or as complex as invoking additional functionality, is a data dictionary (DD) item passed in with the data structure. The DD items appear when the user toggles to personalization mode, and the user can trigger them by choosing the options and then returning to standard mode. If you do not include any such items in the data structure, then personalization mode is disabled for that portlet.

Because the available options are based on the form's data structure, those values are passed in by runtime to the form if they have been set.

11.3.3 Portlet Form Events

Runtime provides three events that are specific to portlet form types:

- **Portlet is Initialized**

This event occurs when runtime initializes the portlet form and before it initializes any forms which the portlet might contain.

- **Personalization Applied**

This event occurs immediately after initialization to apply previously-saved personalization data. It occurs again when the user clicks the Save button in personalization mode.

- **Portlet is Exited**

This event occurs when the portlet is unloaded by the container. Typically, this occurs as the user logs out or the session times out.

In addition to the portlet-specific events, portlets support the general events in this table:

Event	Browse Portlet	Edit Portlet
Grid Record is Fetched	Yes	Yes
Write Grid Line-Before	Yes	Yes
Last Grid Record Has Been Read	Yes	Yes
Notified By Child	Yes	No
Grid Record is Fetched	No	Yes
Write Grid Line-Before	No	Yes
Last Grid Record Has Been Read	No	Yes
Add Record to DB - Before	No	Yes
Add Record to DB - After	No	Yes
Update Record to DB - Before	No	Yes

11.3.4 Edit Portlet Form Design-Time Considerations

In addition to standard form properties, you can control the transaction boundary for edit portlets. You can choose to process each action (that is, insert, update, or delete) on the form separately (**Transaction Disabled**), or you can process all of the actions at once (**Portlet Only Transaction**). Choose the latter if processing each action separately would cause data integrity issues. Transaction processing for edit portlets is identical to that for subforms.

11.4 Generating Portlets (Prior to release 9.2.2)

This section provides an overview of generating portlets and discusses how to deploy an FDA-created portlet.

11.4.1 Prerequisites

Before you complete the tasks in this section:

- Create or obtain a portlet application that was created in FDA.
- Generate the application using eGenerator.

11.4.2 Understanding Portlet Generation

Generating an application based on portlet form types is similar to generating an application based on other form types, the difference being that you must also generate a portlet descriptor file, portlet.xml, that tells the portal which applications are available for the portlet (the WebClient_Portal.war file). This is an overview of the high-level tasks you must complete to create and view a portlet using FDA:

1. Create an application in FDA using the portlet form types.
2. Generate the application (that is, the portlet) with eGenerator.
3. Generate the portlet.xml file into WebClient_Portal.war file with eGenerator.
4. Install the .war file in Collaborative Portal.
5. To view it, add the portlet to a page in the portal.

11.4.3 Deploying an FDA-Created Portlet

This section discusses how to deploy an FDA-created portlet so that you can install it in Collaborative Portal.

To deploy an FDA-created portlet:

1. Launch eGenerator and select Generate, Portlet Deployment.
2. Select the .war file to generate.

Typically, the .war file for the local version of Collaborative Portal resides in C:\B9\system\Generator\WebClient_Portal.war.

eGenerator reads the local specifications for JD Edwards EnterpriseOne and lists all portlets it discovers.

3. Clear the check box next to any portlet that you do not want to generate, and click the Start button.

If multiple versions of an application exist, each version is listed.

4. Select the version you want to generate.

eGenerator creates a new portlet descriptor file and bundles it into a new WebClient_Portal.war file.

11.5 Updating an FDA-Created Portlet after Initial Installation (Prior to release 9.2.2)

This section provides an overview of updating an FDA-created portlet after initial installation and discusses how to:

- Add a new portlet application.
- Delete an existing portlet application.

11.5.1 Understanding Portlet Updates

When you modify an existing FDA-created portlet in FDA, you need only regenerate the JD Edwards EnterpriseOne application to promote the application to the Collaborative Portal. However, if you are removing existing portlets or are adding new FDA-created portlets to Collaborative Portal, you must use eGenerator to update the portlet descriptor, portlet.xml, in the WebClient_Portal.war.

11.5.2 Adding a New Portlet Application

To add a new FDA-created portlet:

1. Launch eGenerator and select Generate, Portlet Deployment.
2. Select the .war file to update.

Select the same war file that you deployed previously.

3. Clear the check box next to any portlets that you do not want to generate, and click the Start button.

You should always select all portlets you want to be deployed to Collaborative Portal because after installation, the new WebClient_Portal.war replaces any previous version that exists on Collaborative Portal server.

eGenerator modifies the portlet descriptor file, portlet.xml, in the .war file.

11.5.3 Deleting an Existing Portlet Application

Regenerating the portlet.xml by removing the portlet definitions does not remove the corresponding portlets from Collaborative Portal when you perform a portlet application update.

Understanding Power Browse Forms

This chapter contains the following topics:

- [Section 12.1, "Power Browse Forms"](#)
- [Section 12.2, "Power Browse Form Hierarchical Structures"](#)
- [Section 12.3, "Power Browse Form Design-Time Considerations"](#)
- [Section 12.4, "Power Browse Events"](#)
- [Section 12.5, "Power Browse Runtime Processing"](#)
- [Section 12.6, "Transaction Boundaries for Power Browse Forms and Subforms"](#)

12.1 Power Browse Forms

Power browse forms are web-only application forms that, through the use of the subform control, enable users to view multiple, interrelated views of data, grids, and tab pages on one form and to pass logic between them. The tab pages can have their own business views (BVs), and these BVs can communicate with each other and can update based on data selection and changes that occur in other BVs on the form. In this way, you can simplify navigation tasks for users.

Power forms have these general properties:

- All regular controls except a parent child control can be placed on a power form.
- Multiple tab controls are permitted.
- The maximize grid feature is supported for all grids.
- All power form (and subform) errors and warnings are shown from the error button on the power form.
- Power forms contain vertical and horizontal scroll bars.
- With Release 9.2.1, power browse forms can be used to create hover forms.

JD Edwards EnterpriseOne offers two types of power forms: browse and edit. Use a power browse form to browse data on a form. You cannot use a power browse form to change data on the form. This form is similar to a find/browse form. A power browse form with a BV should always have a grid with at least one column. You can hide the grid.

See Also: ■[Understanding Find/Browse Forms](#).

12.2 Power Browse Form Hierarchical Structures

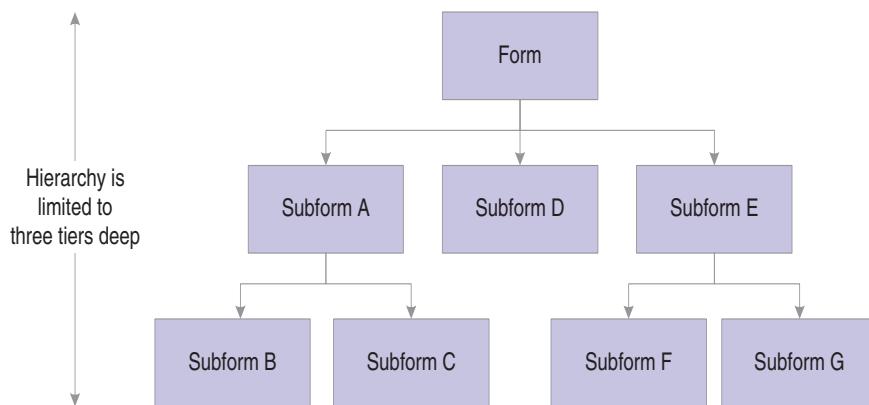
Hierarchical relationships govern the processing between containers in the application. The power form enables you to create hierarchies based on BVs using power forms and subform controls. A power form always resides at the top of the hierarchy. It is a parent form, and can have multiple subform children. Subforms also can function as parent forms, and they can have multiple children. All subforms must have at least one parent power form or subform. At most, you can have three levels, including the initial power form, in a power form hierarchy.

Note: Data flow follows the parenting structure in the logical hierarchy.

A subform can communicate only with its parent or its children. Logic from the event rules (ER) on the subform, or its subcontrols, does not flow to or from the other subforms. Related information does not appear in the available objects.

This flowchart illustrates the general hierarchy scheme of power forms:

Figure 12–1 Power form hierarchical scheme example



Object	Passes Logic To and From
Power Form	Subforms A, D, and E
Subform A	Power Form, Subforms B and C
Subforms B and C	Subform A, Power Form
Subform D	Power Form
Subform E	Power Form, Subforms F and G
Subforms F and G	Subform E, Power Form

12.2.1 Examples of the Logic Flow of Power Forms

This flowchart illustrates the flow of logic between a power form and a business function, and the power form and a form interconnect:

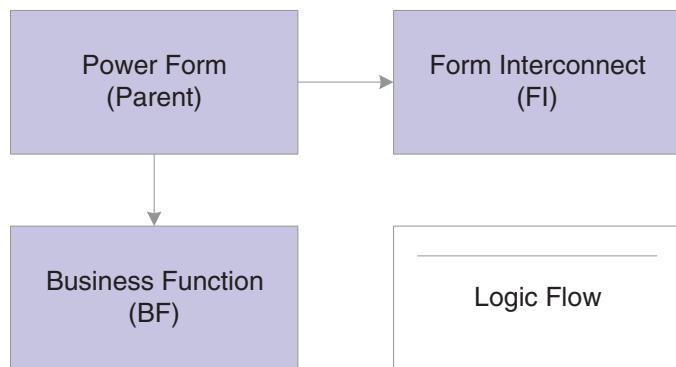
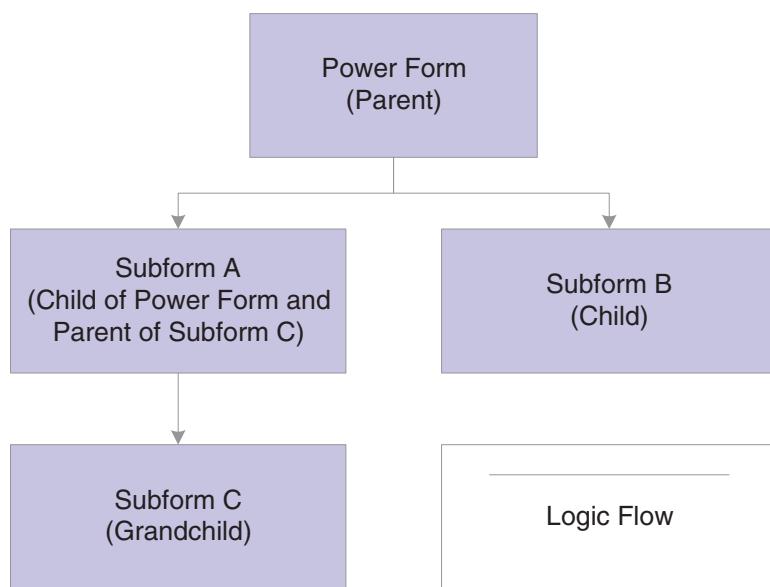
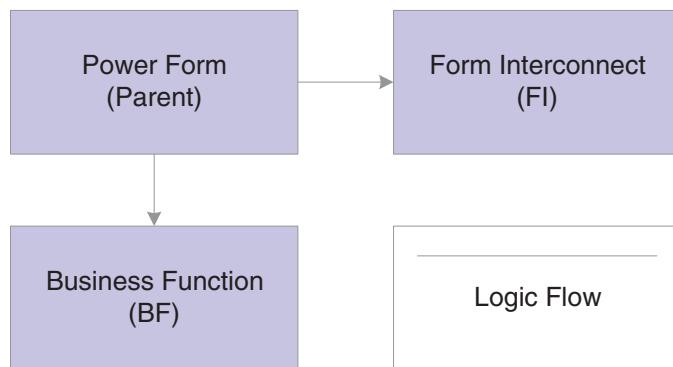
Figure 12–2 Power form logic flow example 1**Figure 12–3 Power form logic flow example 2**

Figure 12–4 Power form logic flow example 3

12.3 Power Browse Form Design-Time Considerations

These property values are particularly significant in the design of power forms:

- Mapping Links
- Transaction
- Hover Form (Release 9.2.1)

Use the Mapping Links property to identify the data to pass between parent and child. For each data item, specify whether the data is to flow from parent to child, child, to parent, or both directions. You must set up a mapping for every child of the parent. Without the mapping, the power browse form probably will not work correctly.

Release 9.2.1

Use the Hover Form property if you would like the form to function as a hover form. When you select the Hover Form property, the other properties within Settings become unavailable. This property replaces using a message form as a hover form.

When a user hovers on an orange dot and clicks the hover indicator in the hover supported control, a pop-up window with context-based information is displayed.

12.4 Power Browse Events

These events can occur on power browse forms during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Write Grid Line-Before
- Write Grid Line-After
- Last Grid Record Has Been Fetched
- Notified by Child
- End Dialog

12.5 Power Browse Runtime Processing

This section discusses how runtime processes power browse forms.

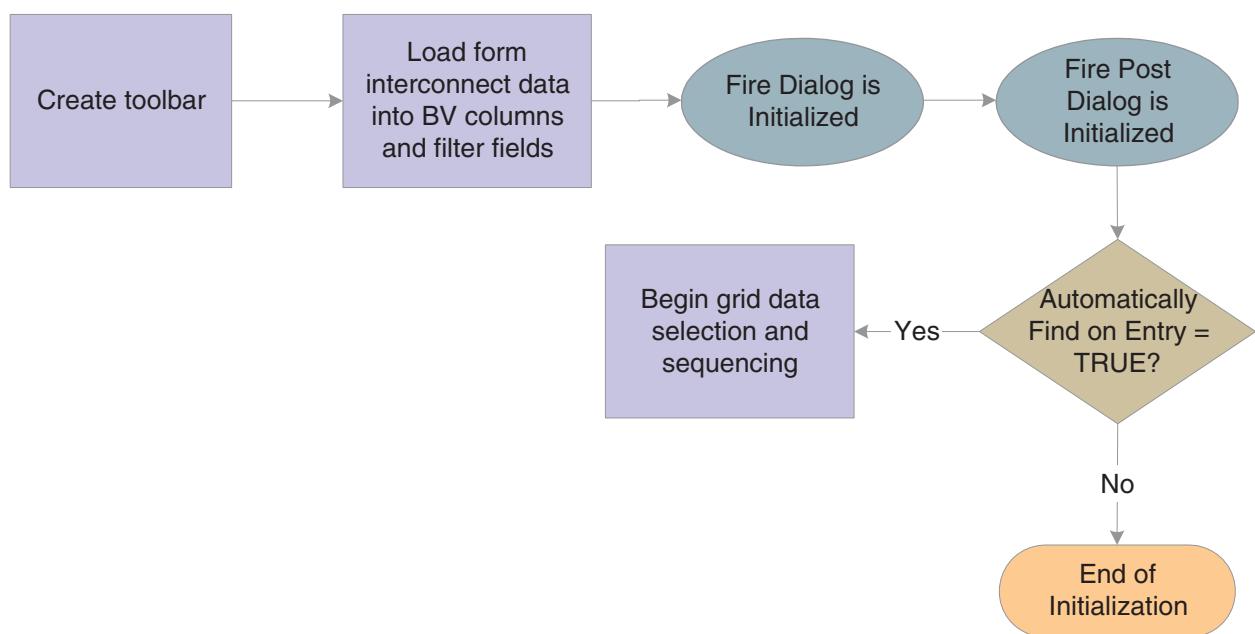
12.5.1 Dialog Initialization

When a power browse form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. BV columns
4. Form controls (FC)
5. Grid fields
6. Static text
7. Helps
8. ER structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization:

Figure 12-5 Power browse dialog initialization



Grid data selection and sequencing occurs at the control level and ultimately leads to the population of the grid, provided runtime encounters no errors.

See [How Runtime Processes the Grid Control](#).

The system creates an internal structure that represents the data selection and data sequencing requirements specified by the user. The system then passes this to the database engine to perform the actual database select and sequencing. The data used for selection is based on values from filter fields and query-by-example (QBE) columns. The system holds the data until the data is retrieved.

12.5.2 Find Button

The Find button is a standard button on power browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the filter fields, runtime performs data selection and sequencing for the grid control. After reloading the grid with the fetched data, runtime fires the **Post Button Clicked** event.

12.5.3 Select Button

The Select button is a standard button on power browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors occur, runtime writes the values from the selected row to the BC and fires the **Post Button Clicked** event. Then it fires the **End Dialog** event and initiates the dialog close process.

12.5.4 Close Button

The Close button is a standard button on power browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime attempts to close all of the modeless child forms, if any exist. If any of these child forms cannot be closed, the Close button process is terminated. Otherwise, runtime fires the **End Dialog** event and initiates the dialog close process.

12.5.5 Dialog Close

Power browse can be closed either by the user (typically by clicking the Select or Close buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BV columns for database commit.
2. Terminate error handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

12.6 Transaction Boundaries for Power Browse Forms and Subforms

Power forms include the Transaction property, which functions the same as the other form types. By default, it is scoped to OK button processing; however, the scope can be extended to business functions, table I/O, and form interconnects.

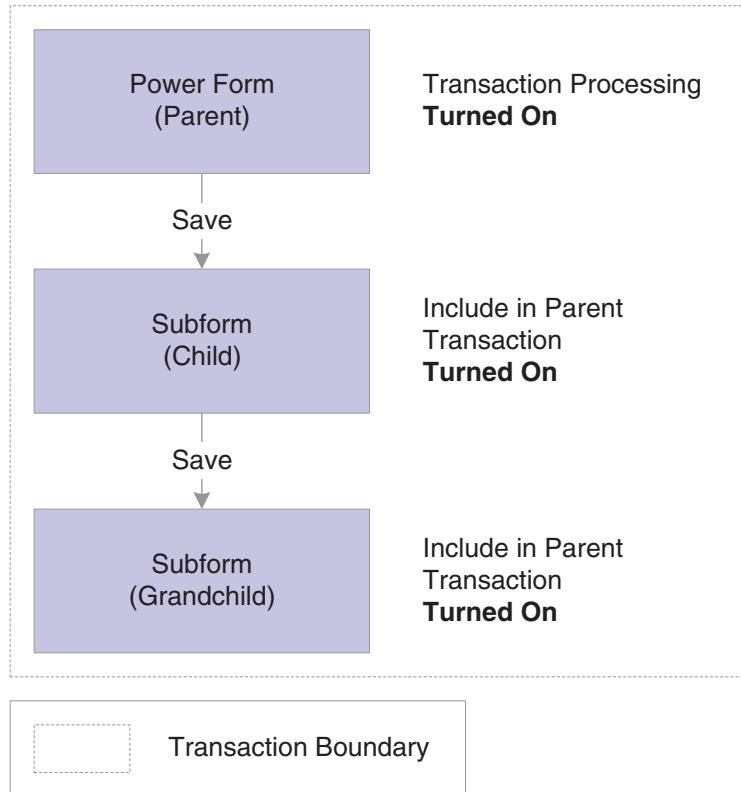
Subforms are scoped to Save button processing; however, the scope can be extended to business functions, table I/O, and form interconnects. You can use these transaction processing settings for subforms:

Setting	Result
Transaction Disabled (default)	No transaction processing occurs for this form.
Include in Parent Transaction	If the form is called within the parent's transaction boundary (OK or Save processing), the transaction will be included in the parent's transaction boundary.

Setting	Result
Subform Only Transaction	Save processing for the Subform has its own transaction boundary, which is independent of all boundaries.

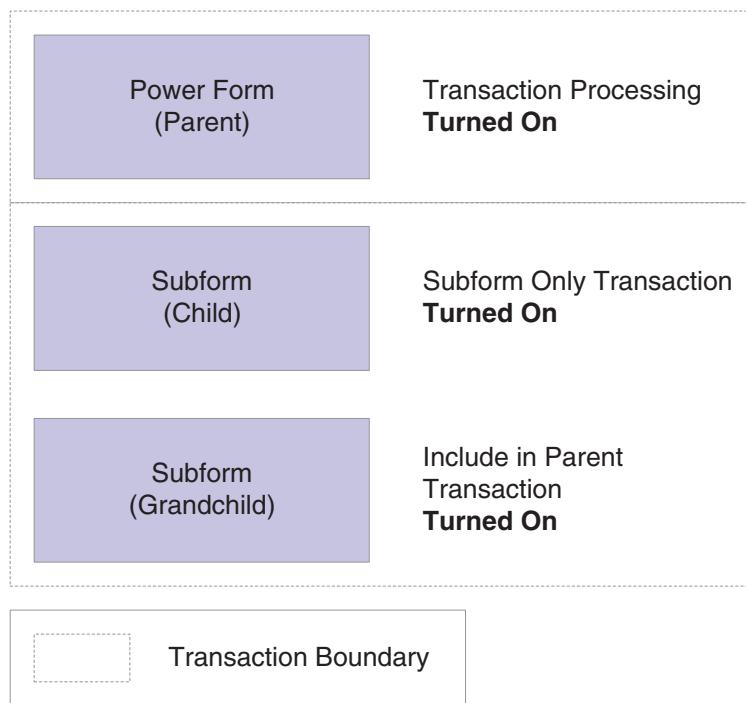
This flowchart illustrates how to include all of the subforms in the form save transaction. If you rollback data on any one form, the system rolls back the changes on all of them:

Figure 12–6 Including parent, child, and grandchild in the transaction boundary



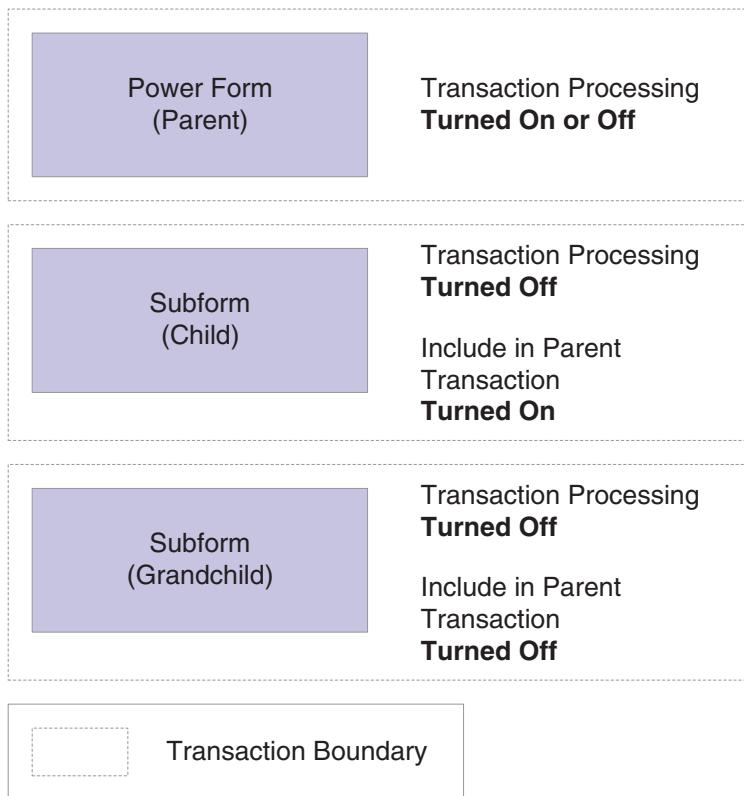
Here, the child is not included in the transaction for the form, which means that a rollback on the parent level will not affect any of the subforms. However, all of the subforms are included in the same transaction, so any rollback on the subforms will affect all of the subforms:

Figure 12–7 Placing parent in a separate transaction boundary from its child and grandchild



In all these scenarios, transactions are local and will not affect appear on any other form:

Figure 12–8 Placing parent, child, and grandchild in separate transaction boundaries



Understanding Power Edit Forms

This chapter contains the following topics:

- [Section 13.1, "Power Edit Forms"](#)
- [Section 13.2, "Power Edit Form Design-Time Considerations"](#)
- [Section 13.3, "Power Edit Events"](#)
- [Section 13.4, "Power Edit Form Runtime Processing"](#)

13.1 Power Edit Forms

Power forms are Web-only application forms that, through the use of the subform control, enable users to view multiple, interrelated views of data, grids, and tab pages on one form and to pass logic between them. The tab pages can have their own business views (BVs), and these BVs can communicate with each other and can update based on data selection and changes that occur in other BVs on the form. In this way, you can simplify navigation tasks for users.

Power forms have these general properties:

- All regular controls except a parent child control can be placed on a power form.
- Multiple tab controls are permitted.
- The maximize grid feature is supported for all grids.
- All power form (and subform) errors and warnings are shown from the error button on the power form.
- Power forms contain vertical and horizontal scroll bars.

JD Edwards EnterpriseOne offers two types of power forms: browse and edit. A power edit form with a grid enables users to update and enter multiple records simultaneously. Similar to a headerless detail form, a power edit form has only one BV.

See Also: ■[Understanding Fix/Inspect Forms](#).

- [Understanding Headerless Detail Forms](#).
- [Understanding Power Browse Forms](#).

13.2 Power Edit Form Design-Time Considerations

These property values are particularly significant in the design of the power edit form:

- Mapping Links
- Transaction

Use the Mapping Links property to identify the data to pass between parent and child. For each data item, specify whether the data is to flow from parent to child, child, to parent, or both directions. You must set up a mapping for every child of the parent. Without the mapping, the power edit form probably will not work correctly.

13.3 Power Edit Events

These events can occur on the power edit form during runtime if the form contains a grid control:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Last Grid Record Has Been Read
- Clear Screen Before Add
- Clear Screen After Add
- Write Grid Line-Before
- Write Grid Line-After
- Post Commit
- Notified by Child
- End Dialog

These events can occur on the power edit form during runtime if the form does not contain a grid control:

- Dialog is Initialized
- Post Dialog is Initialized
- Clear Screen Before Add
- Clear Screen After Add
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Post Commit
- Notified by Child
- End Dialog

13.4 Power Edit Form Runtime Processing

This section discusses how runtime processes power edit forms.

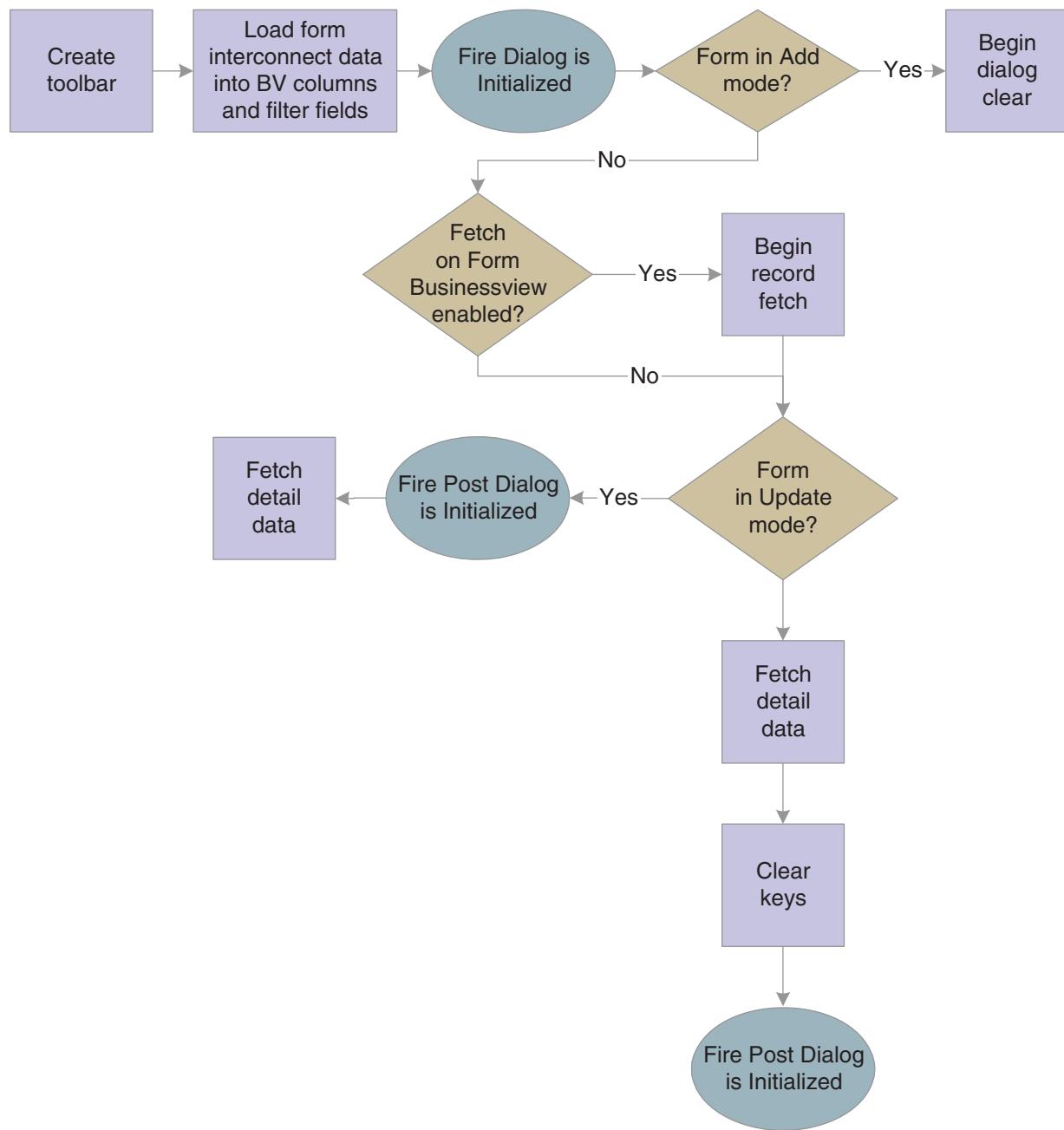
13.4.1 Dialog Initialization

When a power edit form is called, runtime initializes these items in this order:

1. Thread handling

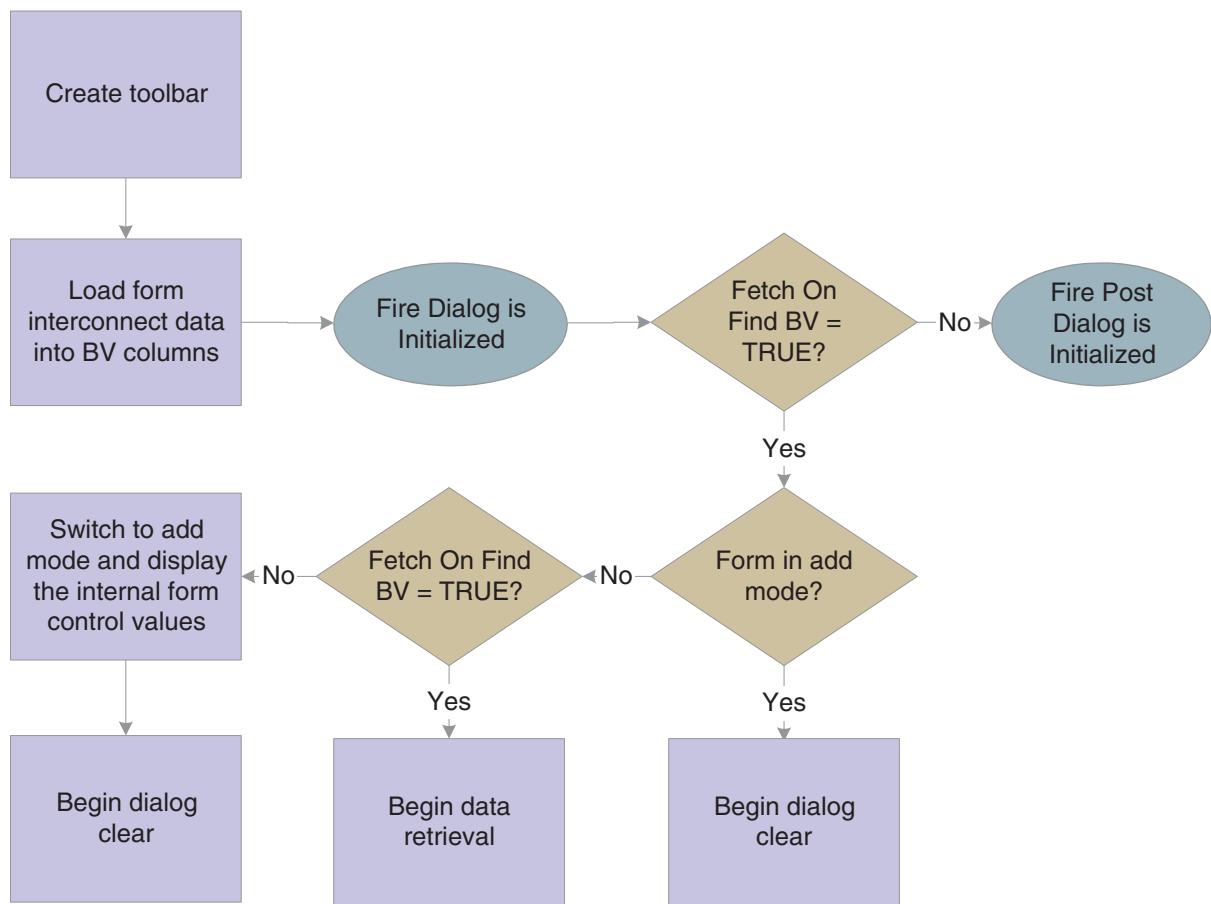
- 2.** Error handling process
- 3.** BV columns
- 4.** Form controls (FC)
- 5.** Grid fields
- 6.** Static text
- 7.** Helps
- 8.** Event rules (ER) structures

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization, if the form contains a grid control:

Figure 13–1 Power edit form with grid control dialog initialization

This flowchart illustrates the tasks that runtime performs after initializing these objects to complete dialog initialization, if the form does not contain a grid control:

Figure 13–2 Power edit form with no grid control dialog initialization



13.4.2 Dialog Clear

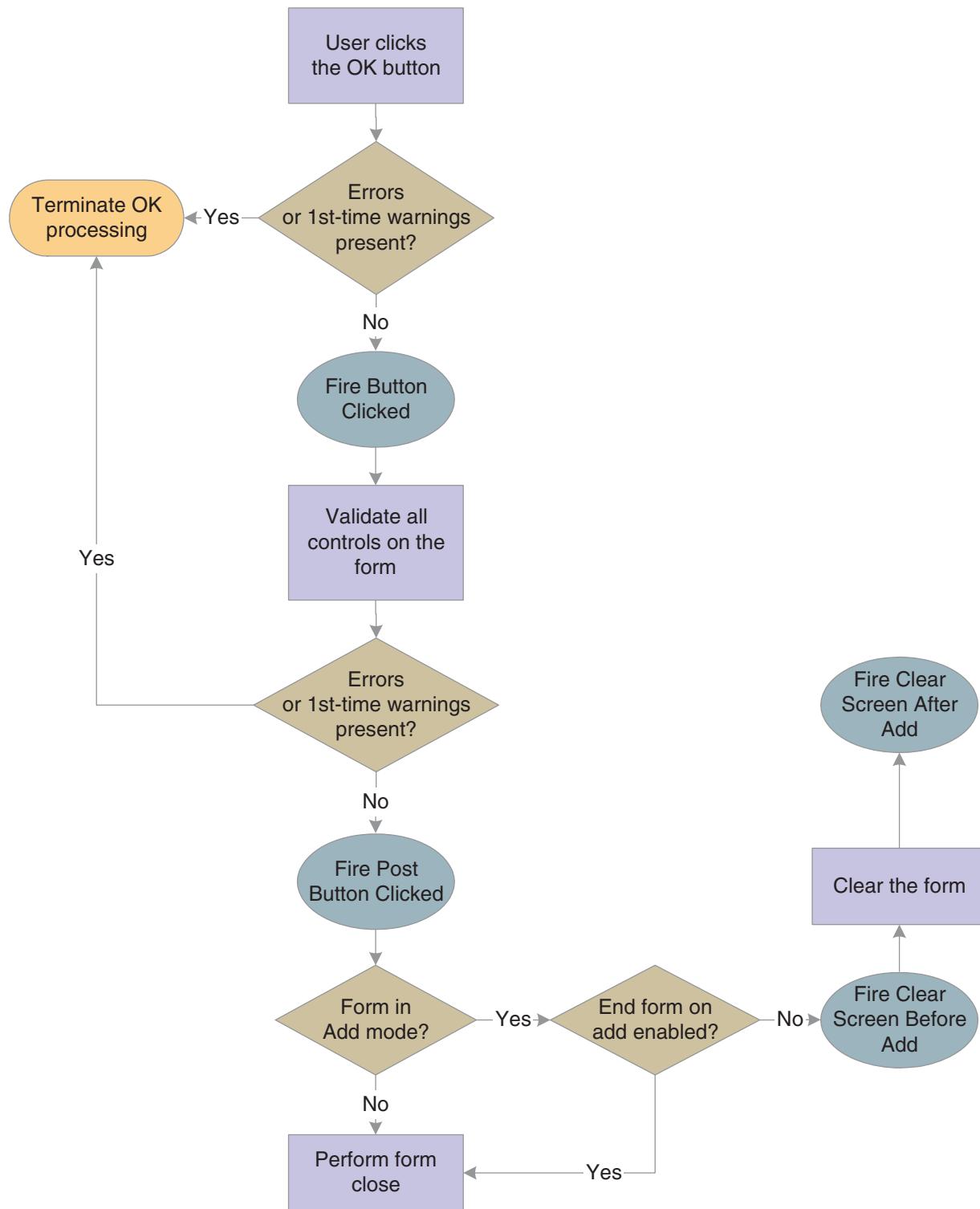
This list discusses how runtime clears the form in preparation to display retrieved data:

1. If the form was called in Copy mode, clear the key (primary index) controls for which the Do not clear after add option was selected.
2. If the form was not called in Copy mode, clear all FCs for which the Do not clear after add option has been selected.
3. Fire the **Clear Screen Before Add** event.
4. Fire the **Post Dialog is Initialized** event.

13.4.3 OK Button

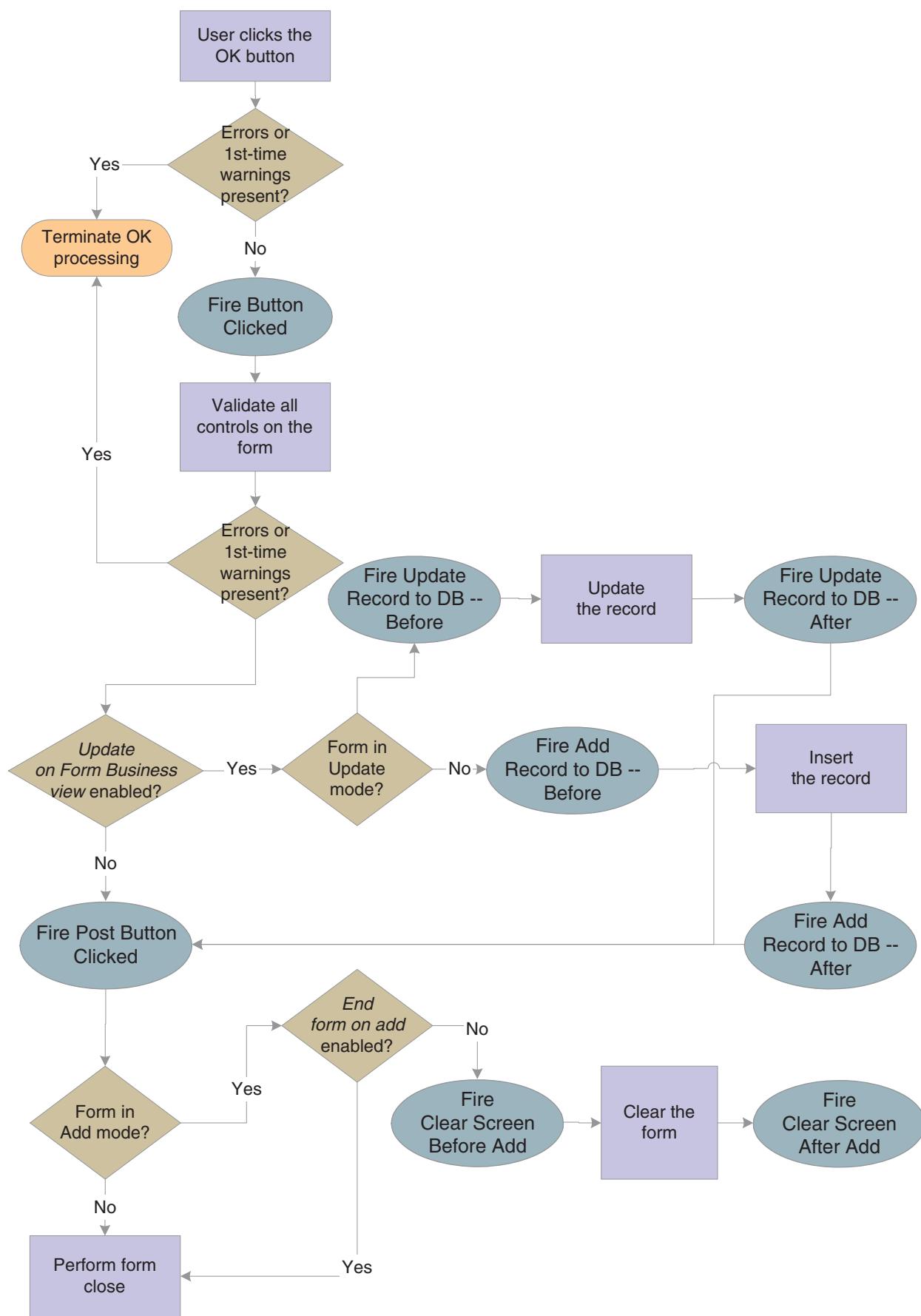
OK is a standard button on power edit forms that appears by default. It causes runtime to validate the information on the form and update or add it to the database through JDEKRNL function calls.

This flowchart illustrates the tasks that constitute runtime processing for the OK button, if the form contains a grid control:

Figure 13–3 Power edit form with grid control OK button processing

This flowchart illustrates the tasks that constitute runtime processing for the OK button, if the form does not contain a grid control:

Figure 13–4 Power edit form with no grid control OK button processing



13.4.4 Cancel Button

The Cancel button is a standard button on power browse forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime cancels any media objects that might be open. Also, if a manual transaction is in process, runtime attempts to cancel it as well. Then, runtime fires the **End Dialog** event and initiates the dialog close process.

13.4.5 Dialog Close

Power edit can be closed either by the user (typically by clicking the OK or Cancel buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BV columns for database commit.
2. Terminate error and thread handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

14

Understanding Search & Select Forms

This chapter contains the following topics:

- [Section 14.1, "Search & Select Forms"](#)
- [Section 14.2, "Search & Select Events"](#)
- [Section 14.3, "Search & Select Runtime Processing"](#)

14.1 Search & Select Forms

A search & select form is used to select a single predetermined field from a record in a predetermined file. Search & Select forms return only one value to the calling form, based on the dictionary alias. If no data dictionary (DD) alias matches the value, the first value from the data structure is returned.

14.2 Search & Select Events

These events can occur on the search & select form during runtime:

- Dialog is Initialized
- Post Dialog is Initialized
- Grid Record is Fetched
- Last Grid Record Has Been Read
- Write Grid Line-Before
- Write Grid Line-After
- End Dialog
- XAPI Subscribe Event

14.3 Search & Select Runtime Processing

This section discusses how runtime processes search & select forms. A search & select form should be attached as a visual assist only to data items that have the same data type as the form data structure element.

This section discusses form-level runtime processing only. Much of the runtime processing for the search & select "form" actually occurs on the level of the grid control, however.

See [How Runtime Processes the Grid Control](#).

14.3.1 Dialog Initialization

When a search & select form is called, runtime initializes these items in this order:

1. Thread handling
2. Error handling process
3. Business view columns (BC)
4. Form controls (FC)
5. Grid fields
6. Static text
7. Helps
8. Event rules (ER) structures

Next, it performs these actions in this order:

1. Create the tool bar.
2. Load form interconnection data into corresponding BC and filter fields, if any.
3. **Fire Dialog is Initialized.**
4. **Fire Post Dialog is Initialized.**
5. If the Automatically Find On Entry option is selected, begin detail data selection and sequencing in the grid control.

If runtime does not encounter errors, this step leads to the population of the grid control, ultimately.

14.3.2 Find Button

The Find button is a standard button on headerless detail forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the filter fields, runtime performs data selection and sequencing for the grid control. After reloading the grid with the fetched data, runtime fires the **Post Button Clicked** event.

14.3.3 Select Button

The Select button is a standard button on search & select forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** event. If no errors occur, runtime writes the values from the selected row to the BC and fires the **Post Button Clicked** event. Then it fires the **End Dialog** event and initiates the dialog close process.

14.3.4 Close Button

The Close button is a standard button on search & select forms that appears by default. When the user clicks it, runtime fires the **Button Clicked** and **Post Button Clicked** events in immediate succession. If no errors occur, runtime attempts to close all of the modeless child forms, if any exist. If any of these child forms cannot be closed, the Close button process is terminated. Otherwise, runtime fires the **End Dialog** event and initiates the dialog close process.

14.3.5 Dialog Close

Search & Select can be closed either by the user (typically by clicking the Select or Close buttons) or by the system. After performing any control-level close processing that might need to occur, runtime closes the form. If the event has not already occurred, runtime fires the **End Dialog** event. Then it performs these tasks in this order:

1. Load form interconnect data from BC for database commit.
2. Terminate error handling.
3. Terminate helps.
4. Free all form structures.
5. Destroy the window.

15

Understanding Wizard Forms

This chapter contains the following topics:

- [Section 15.1, "Wizard Forms"](#)

15.1 Wizard Forms

The wizard form is a simple form with the sole purpose of containing a wizard control. The wizard control is actually the wizard itself, for all intents and purposes. Each wizard control contains multiple pages; each page contains one subform and nothing else. Each subform in the wizard control is parented to this form.

During runtime, the wizard displays the first page and waits for the user to complete the task. Typically, you use the pages to prompt the user to provide input. After completing the task, the user clicks the Next button, and the wizard displays the second page in the list. In a standard scenario, the user continues in this fashion until the last page is reached, at which point the wizard validates and commits the data and then ends.

You cannot attach a business view (BV) to a wizard form. You cannot place any controls except the wizard control on the wizard form. Additionally, the wizard form does not display a tool bar or menus.

The wizard form has no unique property settings. Furthermore, the form does not support event rules (ER); all ER must be placed on the wizard control or its pages instead.

When the wizard runs, the wizard form expands to fill the entire frame. The JD Edwards EnterpriseOne navigation menu is unavailable while the wizard is running. Consequently, the user cannot launch any other JD Edwards EnterpriseOne applications until exiting the wizard.

See Also: ■[Understanding Wizard Controls](#).

16

Understanding External Forms (Release 9.2.1)

This chapter contains the following topics:

- [External Forms Overview](#)
- [External Form Properties](#)
- [Data Structures](#)

16.1 External Forms Overview

An external form is the window through which users interact with an external application like an ADF application, or a JET page or Composed EnterpriseOne Page.

The principle value of an external form is that it provides a direct link to an external application and allows it to be displayed within an EnterpriseOne web client. This means it can be a task, within a Composed EnterpriseOne Page, on a menu, linked to from an EnterpriseOne page, or accessed through FastPath.

Note: With Release 9.2.1.4, you can also use a modal form interconnect or Shortcut Launcher to launch external applications. However, you cannot use Shortcut Launcher to open external forms that are ADF applications.

If using Shortcut Launcher to open an external form, you can see the tasks on the form, but you cannot launch the tasks. (Release 9.2.1.4)

External forms can also be secured just like any other application. You can pass values into the external application using the data structure of the external form. You can also use static text controls on the external form, so when the external form is called using the AIS form service, those static text fields will be returned as translated labels.

Before creating your external form, you must have your external content first, whether that is an ADF application, EnterpriseOne Page/JavaScript, or Composed EnterpriseOne Page. You should also know what kind of input is expected, so you can provide the correct data structure members in the external form. Then you can create the external form in FDA for that content.

Note: You cannot add any controls or events to an external form except static text controls. This is due to the fact that an external form is just a container for the external content to be executed.

16.2 External Form Properties

The external form supports the following properties:

- External Application Type
- External Application
- Entry Point

16.2.1 External Application Type Property

Choose the external application type from the drop-down menu. The valid values are:

- JavaScript - This is a classic EnterpriseOne page or an EnterpriseOne page created with Oracle JavaScript Extension Toolkit (JET). Other javascript libraries may also be used, but JET libraries are provided within EnterpriseOne specifically for use in EnterpriseOne pages.
- Adf - This is an ADF bounded task flow.
- Composed Page - This is a Composed EnterpriseOne Page.

Note: If using JavaScript or Composed Page application types, you must have created these User Defined Objects (UDOs) first so that you have the unique ID to enter in the External Application field.

Also, you must share the page in order for other users to see it. However, you do not need to enable view security for others to see it. Application security is used instead to control who can run and view the external application.

16.2.2 External Application Property

This section discusses valid values for the External Application field.

16.2.2.1 JavaScript

If the external application type is JavaScript, enter the object name of the UDO in the External Application field.

You can find the JET (JavaScript) object name by following these steps:

1. On the EnterpriseOne screen, click the user personal information located on the right side of the banner bar at the top.
2. Under the Personalization category, click the Manage Content link, then click Classic Pages.
3. Select the name of the Classic Page that represents your JET object and click the information icon.

The About form displays the EnterpriseOne Page information including the Object Name. You can copy the Object Name from here.

You can also look up the object name in P98220W.

16.2.2.2 ADF

If the external application type is Adf, enter the bounded task flow ID in the External Application field.

See "Finding the ADF Bounded Task Flow ID" in the *JD Edwards EnterpriseOne Tools Deploying and Developing Oracle Application Development Framework (ADF) Applications for EnterpriseOne* guide for more information.

16.2.2.3 Composed EnterpriseOne Page

If the external application type is Composed Page, enter the object name of the UDO in the External Application field.

You can find the Composed EnterpriseOne Page object name by following these steps:

1. On the EnterpriseOne screen, click the user personal information located on the right side of the banner bar at the top.
2. Under the Personalization category, click the Manage Content link, then click Composed Pages.
3. Select the name of the Composed EnterpriseOne Page that you are looking for and click the information icon.

The About form displays the Composed EnterpriseOne Page information including the Object Name. You can copy the Object Name from here.

You can also look up the object name in P98220W.

Note: Be very careful when entering the external application value as there is no immediate error checking. This field will allow any value and you will not receive an error if the external application does not exist or is mis-typed.

16.2.3 Entry Point Property

External forms can be entry forms. Entry point is an option which, when selected, flags the form as being the one that you want users to see when they first launch the application.

If the form is flagged as an entry point, you can access it directly through FastPath and place it on a menu using only the application name.

Once these forms are configured properly, these forms can be used across EnterpriseOne as normal forms can be, with Form Interconnects, FastPath, and so forth.

16.3 Data Structures

You can define a form data structure for external forms. External form types of JavaScript and ADF both support passing data into the external application via the data structure. Currently, you can only send data into the external form, you cannot receive data values back.

For more information regarding ADF, see Consuming Form Interconnect Values in the *JD Edwards EnterpriseOne Tools Deploying and Developing Oracle Application Development Framework (ADF) Applications for EnterpriseOne* Guide.

For more information regarding JET, see Consuming Form Interconnect Values in the *JD Edwards EnterpriseOne Tools UX One Deployment and Development Guide*

Understanding Calendar Controls

This chapter contains the following topics:

- [Section 17.1, "Calendar Controls"](#)
- [Section 17.2, "Calendar Control Design-Time Considerations"](#)
- [Section 17.3, "Calendar Control Events"](#)
- [Section 17.4, "Calendar Control Runtime Processing"](#)
- [Section 17.5, "Calendar Control System Functions"](#)

17.1 Calendar Controls

With the calendar control, you provide users with a graphical display of a calendar which can show activities (for example, meetings and appointments) and other time sensitive information. It contains three views: day view, week view and month view. It is also fully customizable: you can hide any view, customize work hours and work weeks, and so forth.

The calendar control does not support a business view (BV); hence, it does not fetch or save calendar data. It relies on application logic (BVs and table I/Os) to manage data. However, it does provide a set of event rules (ER) to load, view the details of, and add, delete, or modify calendar activities. An *activity* is a calendaring and scheduling entity that represents a designated amount of time on a calendar. Every calendar event, reminder, and so forth that users or the system schedules is an activity.

Note: In Form Design Aid (FDA), events are points during runtime where you can execute code commands. However, events are also dates that you schedule on a calendar. To avoid confusion, use the term, *activity*, to mean the second kind of event.

The calendar control supports Universal Time (UT) and adjusts for user profile time zone settings. All activities appear based on the user's time zone daylight savings properties in user profile setting. The control itself does not include a function for reminders; therefore, reminder functions must be included as a part of the underlying application itself. The same is true for to-dos and attachments. If the application must display a to-do list, then it is recommended that you use a grid control to do so. The calendar control does not support the ability to attach items to an event directly on the control itself. However, a JD Edwards EnterpriseOne application can associate an attachment with each calendar activity and then display the attachment in the activity details form.

The calendar control does not support mobile devices currently.

The minimum dimensions of the control in FDA are 233 dialogue units high and 428 dialogue units wide. This is to ensure that the control will display correctly on 800 x 600 screen resolution settings. At a 1280 x 960 resolution, you might want a larger calendar control.

The only calendar control-specific design-time property you can set is which views are visible to the user. The calendar control has a variety of specialized system functions and ER.

17.2 Calendar Control Design-Time Considerations

You can choose to hide views from the user with a calendar control property. Alternatively, you can use the system function, **Set View Visible**, to hide and show views during runtime. Because the user must have at least one view to be able to interact with the control, FDA will not permit you to hide all of the views.

17.3 Calendar Control Events

These events, which are unique to calendar controls, can fire on this control:

- Load Calendar Activity
- Drill Into Calendar Activity
- Drill Into Time Span
- Add Activity Button Clicked
- Post Add Activity Button Clicked

17.3.1 Load Calendar Activity

When the form is opened, after the **Post Dialog Is Initialized** form-level event, the **Load Calendar Activity** event fires for all calendar controls on the form. Inside this event, the application can use table I/O or business functions to read calendar data and add the activities to the calendar control for display. The system provides two system variable: SL_StartTime and SL_EndTime. These values define the time range that the calendar control displays. The application logic should fetch all calendar activities that fall between SL_StartTime and SL_EndTime. Both SL_StartTime and SL_EndTime are UTIME type variables.

The **Load Calendar Activity** event fires when the calendar control must display a time period so that the application has an opportunity to load the calendar activities in that time period and to display them on the calendar. For example, when a user navigates to a new day, week, or month, the **Load Calendar Activity** is triggered so that the application can load the activities for the new day, week, or month.

The calendar control provides activity caching. When a day, week, or month is revisited for the second time, the calendar control displays the cached activities for the day, week, or month unless the cache is cleared by the application logic. (One way to clear the calendar cache is to call the system function, **Delete Calendar Activity**, to delete all activities.)

Whenever you want to load the calendar with events that exist in a given time period, use this formula:

```
SELECT from where T2 >= S and T1 <= E
```

where

- T2 is the activity end time,

- S is the system variable, **Slstart**,
- T1 is the activity start time, and
- E is the system variable, **Slend**.

17.3.2 Drill Into Calendar Activity

The calendar control enables a user to click an activity. When that happens, the **Drill Into Calendar Activity** event fires. Consequently, application logic can open a JD Edwards EnterpriseOne form to display the details of the activity for user to modify or delete.

The system outputs a system variable, **SL_CalendarActivityID**, containing the ID of the Activity being clicked.

17.3.3 Drill Into Time Span

The calendar control enables a user to click a time span containing an activity. When that happens, the **Drill Into Time Span** event fires. Consequently, application logic can open a JD Edwards EnterpriseOne form for the user to create a new event. System variables, **SL_StartTime** and **SL_EndTime**, indicate the start and end of the time span the user has clicked.

Double-clicking an empty time span (that is, a time span to which no activity is currently assigned) calls the **Add Calendar Activity** system function.

17.3.4 Add Activity Button Clicked and Post Activity Button Clicked

The calendar control provides an Add button. When this button is clicked, the system triggers the **Add Activity Button Clicked** event and then the **Post Add Activity Button Clicked** event. Applications can use these events to open a JD Edwards EnterpriseOne form for user to create a new activity.

17.4 Calendar Control Runtime Processing

This section discusses runtime processing for calendar controls.

No control-specific runtime processing occurs upon form close.

17.4.1 Initialize the Control

When runtime initializes the calendar control, it loads the control and then fires the event, **Load Calendar Activity**. What runtime loads into the control is based on any system functions defined in ER for this event. The date range that defines the initial load is determined by the view (day, week, or month) that has been set. Keep in mind that while up to three views can be visible, the user can interact with only one at a time—the current calendar view.

Calendar view can be set by the system function, **Select Calendar View**. Alternatively, if only one view is visible because the other two have been hidden either by the system function, **Set View Visible**, or by disabling them at design-time, then runtime sets the one remaining view. In all other scenarios, the order of precedence for the view to set is day, week, then month.

Runtime sets the start and stop system variables (**SL_StartTime** and **SL_EndTime**, respectively) based on the current view. Then runtime passes the variables to **Load Calendar Activity**, and the system in turn populates the current view of the control. Only the activities for the current view are loaded. The control populates activities for

other dates and views as the user travels to them.

17.4.2 Add a Calendar Activity

This scenario occurs most commonly as a result of control initialization or because a user travels to a date that he has not traveled to before during the course of the session. A user can travel to a new date by clicking the Next or Previous buttons.

Upon the button click, runtime uses ER to call the system function, **Load Calendar Activity**, with the start and end time being passed in as system values. Runtime derives the start and end times from the new date to which the user has just traveled. The system then populates the control with the newly fetched data.

After the initial fetch, data for each date is kept in memory to avoid unnecessary data refreshes. That is why runtime calls **Load Calendar Activity** only when the user travels to a date that has not been loaded for the current session.

17.4.3 Refresh the Control

Sometimes, you might want to force a refresh. To do so, call the system function, **Delete Calendar Activity**, and pass in **Delete All Activities** for the **Activity ID** parameter. Runtime responds by removing all activities from the entire control and flushing the memory where previously traveled-to dates are stored. Runtime also sets the start and stop system variables to match the view that is currently set. In other words, runtime places the control in a condition similar to being newly initialized.

After calling **Delete Calendar Activity**, call **Load Calendar Activity**, passing in the start and stop system variables (SL_StartTime and SL_EndTime) to populate the current control view.

17.5 Calendar Control System Functions

This section discusses the system functions unique to the calendar control.

In general, you can use system functions to perform these types of tasks:

- Add, modify, or delete an activity from the calendar control.
- Classify time ranges as being working, nonworking, or of another special type.
- Modify aspects of the calendar control interface.

When you add an activity through the system function, **Add Calendar Activity**, or modify an activity using the system function, **Modify Calendar Activity**, you can affect these aspects of how the activity appears and acts on the calendar:

- Name of the activity (subject).
- Start and end day and time.

The system requires this value in JDEUTime, even when you designate an activity as an all-day activity. You can also flag an activity for reoccurrence.

If the start and stop times result in a span of time that is greater than or equal to 24 hours, then the system converts the activity to an all day activity. All day activities are displayed in the all day event area in the interface.

- Additional information associated with the activity.

You can set values for the type (meeting, appointment, and so forth), the commitment level (low, medium, or high), the activity location, and the activity organizer.

- Private.

This option causes the system to display a special icon next to the activity. No other logic is applied automatically, however. If you wish to restrict access to an event marked private, then you must do so within the application.

- Font, color, and icons.

You can set the font and related attributes, set a background color, and associate an icon with any activity. The icons from which you have to choose must already reside in the system. To add an icon to the list, save a GIF-type graphic in both of these locations: the B9\<pathcode>\res folder on the machine where FDA is located and the webclient.war/img/res folder on the machine or machines hosting the web server.

When modifying an event, ensure that the ID value for the event exactly matches the ID as it appears in the table. For example, if the table column is left-padded, you must prefix the event ID as it appears in the control with the same number of characters that have been added to the event ID in the table because of the column formatting.

When you delete an activity using the system function, **Delete Calendar Activity**, you remove it from the calendar control; you do not remove it from the database. You can choose to delete a single activity or multiple ones. You can also choose to clear the control of all activities.

In some cases, it might be helpful to assign different classifications to different date ranges. Typical classifications include working and nonworking (for example, holidays). Use **Set Work Day Hours** and **Set Work Week** to designate work time. Currently, **Set Work Day Hours** only affects full hours; it cannot accept half-hour increments, for example.

Furthermore, the work week must not contradict the work week system settings for the current country. For example, in the US, work weeks cannot end on Sunday or start on Saturday. Additionally, a work week cannot span these start and stop days. For example, you cannot start a work week on a Tuesday and end on a Monday.

Use **Set Day Type** to classify days as belonging to any other special categories.

The calendar control can provide users with three views: day, week, and month. Use **Select Calendar View** to display the calendar in a specific view type. You can also prevent users from being able to see a particular view type with **Set View Visible**.

The calendar control also provides an Add button. To change the text on the Add button, use **Set Add Button Text**. To hide the Add button and therefore prevent users from adding their own activities, use **Set Add Button Visible**.

Note: The Add button has no underlying logic. If you choose to display the button, you should assign it some function on the **Add Activity Button Clicked** event.

Add Calendar Activity

Use this system function to add one activity to the calendar control.

Parameters

Calendar Control

Input, required. The calendar form control (FC) to affect.

Activity ID

Input, required. The ID of the activity in the database to be added to the calendar. Set the parameter to an applicable object from the object list.

Subject

Input, required. The title of the activity as it should appear on the calendar for the user. Set the parameter to an alphanumeric constant (**<Literal>**) or an applicable object from the object list.

Start Date and Time

Input, required if All Day Activity = False. The starting time and date, in UTime, for the activity. Literal date value in JDEUTime format.

End Date and Time

Input, required if All Day Activity = False. The ending time and date, in UTime, for the activity. Literal date value in JDEUTime format.

All Day Activity

Input, required. The indicator of whether the activity is to be displayed as an all day activity instead of having start and end date and times. Any activity that is equal to or greater than 24 hours automatically becomes an all day activity. All day activities are displayed in the all day activity area in the interface. Set the parameter to **<TRUE>**, **<FALSE>**, or an applicable object from the object list.

Recurrence

Input, required. The indicator of whether the activity occurs more than once. This parameter merely sets a flag; the control displays an icon to indicate that the activity has been flagged as reoccurring, but the system performs no other action in connection with recurrence. Set the parameter to **<TRUE>**, **<FALSE>**, or an applicable object from the object list.

Type

Input, required. The classification of the activity (meeting, appointment, and so forth). This parameter does not appear on the calendar. Set the parameter to an alphanumeric constant (**<Literal>**) or an applicable object from the object list.

Commitment Level

Input, required. The relative value of the activity. The value assigned affects the icon displayed and the color of the duration bar for the activity. Set the parameter to **<Low Commitment>** (1), **<Default>** (2), or **<High>** (3).

Private

Input, required. The indicator of whether to display the "private" icon. Set the parameter to **<TRUE>**, **<FALSE>**, or an applicable object from the object list.

Location

Input, required. The name of the place where the activity is to occur as it should appear on the calendar for the user. Set the parameter to an alphanumeric constant (<Literal>), <N/A>, or an applicable object from the object list.

Organizer

Input, required. The name of the creator of the activity as it should appear on the calendar for the user. Set the parameter to an alphanumeric constant (<Literal>), <N/A>, or an applicable object from the object list.

Font

Input, required. The font to use to display the activity to the user. Set the parameter to a font and related settings from the Font dialog (<Pick Font>) or the default font and related settings (<Reset Font>).

Bitmap

Input, required. The bitmap of an icon to display to the user on the calendar in association with the activity. To add an icon to the list, save a GIF-type graphic in both of these locations: The B9\<pathcode>\res folder on the machine where FDA is located and the webclient.war/img/res folder on the machine or machines hosting the Web server. Set the parameter to a bitmap from the system-supplied list (<Choose Calendar Bitmap>) or the default icon (<Default Calendar Bitmap>).

Background Color

Input, required. The color to appear behind the information on the calendar. Set the parameter to a color from the color palette (<Pick Color>) or the default color (<Reset Color>).

Additional Notes

When you add an activity through the system function, **Add Calendar Activity**, you can affect these aspects of how the activity appears and acts on the calendar:

- Name of the activity (subject).
- Start and end day and time.

The system requires this value in JDEUTime, even when you designate an activity as an all-day activity. You can also flag an activity for reoccurrence.

If the start and stop times result in a span of time that is greater than or equal to 24 hours, then the system converts the activity to an all day activity. All day activities are displayed in the all day event area in the interface.

- Additional information associated with the activity.

You can set values for the type (meeting, appointment, and so forth), the commitment level (low, medium, or high), the activity location, and the activity organizer.

- Private.

This option causes the system to display a special icon next to the activity. No other logic is applied automatically, however. If you wish to restrict access to an event marked private, then you must do so within the application.

- Font, color, and icons.

You can set the font and related attributes, set a background color, and associate an icon with any activity. The icons from which you have to choose must already reside in the system. To add an icon to the list, save a GIF-type graphic in both of

these locations: the B9\<pathcode>\res folder on the machine where FDA is located and the webclient.war/img/res folder on the machine or machines hosting the web server.

Whenever you want to load the calendar with events that exist in a given time period, use this formula:

```
SELECT from where T2 >= S and T1 <= E
```

where

- T2 is the activity end time,
- S is the system variable, **S1start**,
- T1 is the activity start time, and
- E is the system variable, **S1end**.

See Also: [Modify Calendar Activity](#)

Delete Calendar Activity

Use this system function to delete an existing activity from the calendar control.

Parameters

Calendar

Input, required. Control: The calendar FC to affect.

Activity ID

Input, required. The ID of the activity to be removed from the calendar. Set the parameter to a parameter to delete all events currently loaded for the control (<Delete All Activities>) or an applicable object from the object list.

Additional Notes

When you delete an activity using this system function, you remove it from the calendar control; you do not remove it from the database. You can choose to delete a single activity or multiple ones. You can also choose to clear the control of all activities.

Modify Calendar Activity

Use this system function to modify an existing event on the calendar control.

Parameters

Calendar

Input, required. The calendar FC to affect.

Activity ID

Input, required. The ID of the activity to be changed on the calendar. Ensure that this value exactly matches the ID (including padding and so forth) as it appears in the table. Set the parameter to an applicable object from the object list.

Subject

Input, required. The title of the activity as it should appear on the calendar for the user. Set the parameter to the current value (<No Change>), an alphanumeric constant (<Literal>) or an applicable object from the object list.

Start Date and Time

Input, required if All Day Activity = False. The starting time and date, in JDEUTime, for the activity. Set the parameter to the current value (<No Change>) or a literal date value in JDEUTime format.

End Date and Time

Input, required if All Day Activity = False. The ending time and date, in JDEUTime, for the activity. Set the parameter to the current value (<No Change>) or a literal date value in JDEUTime format.

All Day Activity

Input, required. The indicator of whether the activity is to be displayed as an all day activity instead of having start and end date and times. Any activity that is equal to or greater than 24 hours automatically becomes an all day activity. All day activities are displayed in the all day activity area in the interface. Set the parameter to the current value (<No Change>), <TRUE>, <FALSE>, or an applicable object from the object list.

Recurrence

Input, required. The indicator of whether the activity occurs more than once. This parameter merely sets a flag; the control displays an icon to indicate that the activity has been flagged as reoccurring, but the system performs no other action in connection with recurrence. Set the parameter to the current value (<No Change>), <TRUE>, <FALSE>, or an applicable object from the object list.

Type

Input, required. The classification of the activity (meeting, appointment, and so forth). This parameter does not appear on the calendar. Set the parameter to the current value (<No Change>), an alphanumeric constant (<Literal>) or an applicable object from the object list.

Commitment Level

Input, required. The relative value of the activity. The value assigned affects the icon displayed and the color of the duration bar for the activity. Set the parameter to the current value (<No Change>), <Low Commitment> (1), <Default> (2), or <High> (3).

Private

Input, required. The indicator of whether to display the "private" icon. Set the parameter to the current value (<No Change>), <TRUE>, <FALSE>, or an applicable object from the object list.

Location

Input, required. The name of the place where the activity is to occur as it should appear on the calendar for the user. Set the parameter to an alphanumeric constant (<Literal>), <N/A>, or an applicable object from the object list.

Organizer

Input, required. The name of the creator of the activity as it should appear on the calendar for the user. Set the parameter to the current value (<No Change>), an alphanumeric constant (<Literal>), <N/A>, or an applicable object from the object list.

Font

Input, required. The font to use to display the activity to the user. Set the parameter to the current value (<No Change>), a font and related settings from the Font dialog (<Pick Font>) or the default font and related settings (<Reset Font>).

Bitmap

Input, required. The bitmap of an icon to display to the user on the calendar in association with the activity. To add an icon to the list, save a GIF-type graphic in both of these locations: The B9\<pathcode>\res folder on the machine where FDA is located and the webclient.war/img/res folder on the machine or machines hosting the Web server. Set the parameter to the current value (<No Change>), a bitmap from the system-supplied list (<Choose Calendar Bitmap>) or the default icon (<Default Calendar Bitmap>).

Background Color

Input, required. The color to appear behind the information on the calendar. Set the parameter to the current value (<No Change>), a color from the color palette (<Pick Color>) or the default color (<Reset Color>).

Additional Notes

When you modify an activity using the system function, **Modify Calendar Activity**, you can affect these aspects of how the activity appears and acts on the calendar:

- Name of the activity (subject).
- Start and end day and time.

The system requires this value in JDEUTime, even when you designate an activity as an all-day activity. You can also flag an activity for reoccurrence.

If the start and stop times result in a span of time that is greater than or equal to 24 hours, then the system converts the activity to an all day activity. All day activities are displayed in the all day event area in the interface.

- Additional information associated with the activity.

You can set values for the type (meeting, appointment, and so forth), the commitment level (low, medium, or high), the activity location, and the activity organizer.

- Private.

This option causes the system to display a special icon next to the activity. No other logic is applied automatically, however. If you wish to restrict access to an

event marked private, then you must do so within the application.

- Font, color, and icons.

You can set the font and related attributes, set a background color, and associate an icon with any activity. The icons from which you have to choose must already reside in the system. To add an icon to the list, save a GIF-type graphic in both of these locations: the B9\<pathcode>\res folder on the machine where FDA is located and the webclient.war/img/res folder on the machine or machines hosting the web server.

When modifying an event, ensure that the event's ID value exactly matches the ID as it appears in the table. For example, if the table column is left-padded, you must prefix the event ID as it appears in the control with the same number of characters that have been added to the event ID in the table because of the column formatting.

Whenever you want to load the calendar with events that exist in a given time period, use this formula:

```
SELECT from where T2 >= S and T1 <= E
```

where

- T2 is the activity end time,
- S is the system variable, **S1start**,
- T1 is the activity start time, and
- E is the system variable, **S1end**.

[Add a Calendar Activity](#)

Select Calendar View

Use this system function to determine which view of the calendar (day, week, or month) to display to the user upon an event trigger.

Parameters

Calendar

Input, required. The calendar FC to affect.

Calendar View

Input, required. The calendar view (day, week, or month) to display to the user. Set the parameter to <Day View> (0), <Week View> (1), or <Month View> (2).

Select Calendar View

Understanding Check Box Controls

This chapter contains the following topics:

- [Section 18.1, "Check Box Controls"](#)
- [Section 18.2, "Check Box Control Design-Time Considerations"](#)
- [Section 18.3, "Check Box Events"](#)

18.1 Check Box Controls

Use check boxes to indicate choices. Selecting a check box indicates that the associated function is to be enabled. Check boxes are not mutually exclusive. You must associate a check box with a database or data dictionary (DD) item. You can use a check box to pass a value to an event rule (ER).

18.2 Check Box Control Design-Time Considerations

In addition to the standard control property options, check boxes have two unique properties: Checked Value and Unchecked Value. The control returns one of these values, depending on its state. The default values are 1 for selected (checked) and 0 for cleared (unchecked). You can set the values to any numeric value.

18.3 Check Box Events

The only event that can occur on a check box is **Selection Changed**. Runtime fires this event whenever the user selects or clears the check box, so it can occur at any point during runtime processing that the user interacts with the application.

Understanding Combo Box Controls

This chapter contains the following topics:

- [Section 19.1, "Understanding Combo Box Controls"](#)
- [Section 19.2, "Loading Combo Box Controls"](#)
- [Section 19.3, "Combo Box Control Design-Time Considerations"](#)
- [Section 19.4, "Combo Box Control Events"](#)
- [Section 19.5, "Combo Box Control Runtime Processing"](#)
- [Section 19.6, "Combo Box Control System Functions"](#)
- [Section , "Embedded Combo Box System Functions"](#)

19.1 Understanding Combo Box Controls

You can use a combo box (drop-down list) to display a list of items from which the user can make a selection. The combo box includes a type-ahead feature where typing the first character of a matching item description will select that item in the control. It can reside on a form or inside a grid control or in the grid of a parent child control. When inside a grid control, the combo box acts exactly as a normal grid cell except when it has the focus. At that point, it behaves as a combo box; that is, the user can pick an item from the list by clicking and choosing an item from the drop-down, by typing directly in the field, or by using the arrow and enter keys on the keyboard.

You must associate the combo box with a data dictionary (DD) item. If the associated DD item has user-defined code (UDC) values, those values are used to load the combo box list. During runtime, you can load the control with values from a different UDC, if necessary. Alternatively, you can use event rules (ER) in the application to load the values without reference to a UDC, although doing so precludes the ability to directly include translated text in the drop-down list. In either case, using a system function to remove items, you can dynamically filter the list before it is displayed to the user.

No matter what is loaded into the control, upon initialization, the control always displays -- Select One -- as the currently selected item to indicate that the user should select an item in the control. This is true of combo boxes both on a form and in a grid. Although it is an item in its own right, you cannot remove it from the list. --Select One-- always appears at the top of the list; you cannot insert a list item above it.

Note: -- Select One -- is merely a prompt-an indication to the user to make a selection. --Select One -- is not an actual value. Therefore, it will never be written to the database with the intent of saving it as actual data.

19.2 Loading Combo Box Controls

This section provides an overview of loading combo box controls, and discusses how to:

- Load a Combo Box from a UDC
- Load a Combo Box from Cache
- Load a Combo Box with the Add Item System Function

19.2.1 Loading Combo Box Fundamentals

You can load a combo box control from any of these data sources:

- Its associated DD UDCs.
- Cache (using the **Load from Cache** system function).
- Any source available to the **Add Item** system function.

In all cases, runtime checks each item for valid Key and Item parameter pairs. If the Key is null, or its matching description is null or an empty string, runtime throws an error and does not load that item. Also, to avoid duplicate items from appearing in the list, runtime does not load an item if it finds the item's Key parameter value in the combo box's current item list. This check avoids duplicate list items; however, it does not prevent two (or more) different list items with the same name (Description parameter) from appearing in the list.

No matter what method you choose to load the control, ensure that the user has at least one option to select. As part of the initialization process, if runtime cannot find at least one value to put in the drop-down list, it disables the combo box control until one or more values are loaded into the control.

In some cases, the values to load into the combo box might include an item where the Key is " " (single blank) and the description is "(None)." In fact, in JD Edwards EnterpriseOne applications, this value has been used as a blank default value, historically. Although this is a valid item to load into the combo box, it is recommended that you ascribe a meaningful text string to such an item that makes sense in the context of the application as opposed to descriptions such as (None), or " ."

Note: As of release 8.94, the practice of the runtime engine automatically loading a combo box that is configured in Form Design Aid (FDA) as being not required with a default combo box item of key = " " (single blank) and description = "(None)" is discontinued. If such an item must be loaded in the context of the application in question, that task is now the responsibility of the application developer to define and insert into the control.

19.2.2 Loading a Combo Box from a UDC

The combo box control supports string and numeric data types for the combo box's keys and the matching descriptions are always treated as strings. Dynamic changes to a combo box based on a UDC can be made with the system functions **Set Data Dictionary Item** or **Set Data Dictionary Overrides** to load a new set of key (value)/description pairs.

If the application must support multiple languages, loading from a UDC is probably an ideal choice because of the way JD Edwards EnterpriseOne applies alternate UDC lists automatically based on the user's language code. An alternative to dynamically

changing the values displayed in a combo box at runtime is to use the **Set Data Dictionary Item** or **Set Data Dictionary Overrides** system functions to load the correct set of values as needed. You can suppress inappropriate list items within an existing combo box item list using the Remove Item system function. If you must set up the combo box based on a non-UDC DD item but translated descriptions still need to be displayed to the user, you have some options.

You can use the ER system value, **SL LanguagePreference**, to acquire user's language preference. Use that value to fetch and load items into the combo box manually from the appropriate UDC table (F0005 standard UDC table, or F0005D translated UDC table).

Alternatively, you can define the necessary description items as text variables in the application and then use these text variables in conjunction with the **Add Item** system function to add the items to the combo box.

Runtime always loads from the UDC if one is associated with the combo box's data dictionary item. If the combo box is loaded from a UDC and you want to load from a non-UDC source instead, use **Set Data Dictionary Item** to associate a data dictionary item that does not have a UDC with the combo box. Then you can load from cache or use **Add Item** instead.

19.2.3 Loading a Combo Box from Cache

The **Load From Cache** system function enables you to load a combo box from a JD Edwards EnterpriseOne cache data source. Runtime will load values from cache only if the target combo box control is based on a valid DD item that does not have a UDC list associated with it.

When working from cache, you must have a prepared data structure to govern loading data from source into the cache and then transferring it from the cache to the combo box control. When you choose (or create) a data structure to use for this purpose, ensure that the Key, Item, and optional Filter Column parameter data types are compatible with the corresponding data types demanded by the system functions you will call.

Note: All of the values are retrieved from cache prior to filtering. Therefore, you should avoid loading large quantities of data into cache to reduce the possibility of poor performance.

The JD Edwards EnterpriseOne cache from which the combo box item data will be read must have defined a distinct column containing one or more rows of data for each of these combo box data items:

- E1 Cache Column - Keys (values)
- E1 Cache Column - Items (descriptions)
- E1 Cache Column - Filter Column Values (optional filter value)

If the optional filter cache column is not designated or does not contain valid data, runtime will retrieve and load into the combo box all of the key/description pairs currently loaded in the specified cache. After the combo box has been loaded from the cache, you can dynamically remove items from its item list using the **Remove Item** system function.

19.2.4 Loading a Combo Box with the Add Item System Function

To load a small number of items into a combo box, you can use the **Add Item** system function. If translated description text is needed, you can use text variables for the item descriptions or fetch the data manually from the appropriate UDC table (F0005, or F0005D) based on the user's language preference that is accessible through the system value, SL LanguagePreference.

See Also: ■[Combo Box Control System Functions](#).

19.3 Combo Box Control Design-Time Considerations

The combo box control has no unique property settings. Among the more standard property settings, the Required Field flag deserves mention in relationship to the combo box control. Recall that when initialized, the control simply displays the user prompt, -- Select One --. If the Required Field flag is enabled and this prompt is displayed during validation, the engine throws an error and prevents the user from continuing until he or she selects an item from the list. On the other hand, if the flag is disabled under the same circumstance, the engine will actually write a blank (that is, key =) to the database if the -- Select One -- prompt is still selected at the time the OK or Save button is clicked. The blank is written to the database in this case because the system cannot save an empty string () as a value for a key.

A second consideration when using the Required Field flag with combo boxes is that, if the required field indicator should be displayed in the application, you must check the Required Field property for the combo box and then connect static text to it. Be sure to size the connected static text box appropriately to allow for the "required" indicator to be appended to the end of the text, and also to allow for translations. If you do not connect static text to the combo box, runtime does not display the required field indicator in the application.

To connect static text to the combo box, select the combo box and the label. Then select Edit, Reconnect. To disconnect the static text from the combo box, select them and then select Edit, Disconnect.

19.4 Combo Box Control Events

These events can fire on the combo box control during runtime:

- Selection Changed
- Control is Entered
- Control is Exited

Runtime fires the **Selection Changed** event when the user or application selects an item in the combo box list that is different from the item currently selected. In Microsoft Internet Explorer (IE), this selection can be made using any of these methods:

- User selects an item with the mouse from the combo box's drop-down list.
With the drop-down list displayed, the user can also type the first character of a matching item, or use the up and down arrow keys.
- User places the focus on the combo box without activating its drop-down list and types the first character matching that of an item description in the combo box's list of items (type-ahead feature).

If more than one item that matches this character exists, the user can cycle through all of the matching items by pressing the same character key multiple times. Each time a new item is selected, the **Selection Changed** event fires.

- User places focus on the combo box without activating its drop-down list and then uses the up and down arrow keys to cycle through the combo box's item list until the desired item is found.
Each time a new item is selected, the **Selection Changed** event fires.
- Combo box is the target of an ER assignment statement and the value assigned to it is a valid key value matching one of the current item key values loaded in the control.
- Combo box and a valid key value are passed into the combo box system function, **Select Item**.

In some explorers other than IE (such as Mozilla), the user might need to tab out of the combo box to trigger the **Selection Changed** event, particularly if the selection was made using the keyboard.

Of course, runtime fires **Control is Entered** when the user changes the focus to the check box by any of these means, and fires **Control is Exited** when the control loses focus.

19.5 Combo Box Control Runtime Processing

This section discusses how the runtime engine processes the combo box.

19.5.1 Control Initialization

When runtime loads the control, items with a blank key are valid, but display no description if the description parameter is also blank. If the item has a key which is a string of one or more continuous blank characters, runtime concatenates the multiple blank characters to a single blank character. Similarly, the engine removes blank characters that follow an otherwise non-blank item key value before inserting it into the control.

Note: The previous behavior of the HTML runtime engine automatically loading a combo box item having:

key = "" (an empty string) or " " (blank) (depending on SP level)

and

item description = (None)

when the control is marked in FDA as being not required is discontinued as of JD Edwards EnterpriseOne 8.94.

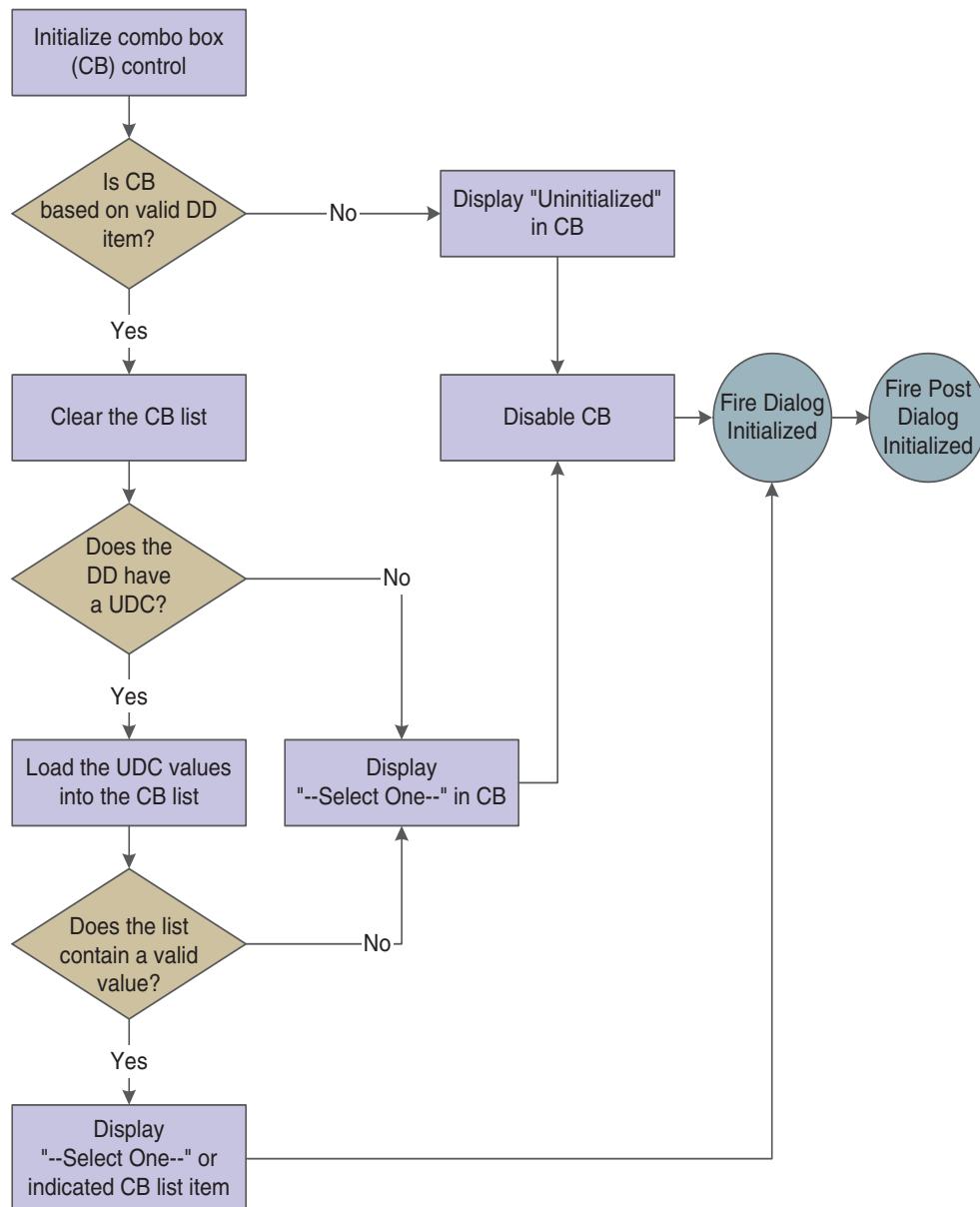
Additionally, to be functional, a combo box must have a valid DD item associated with it. Combo boxes that do not have such an associated item are rendered displaying the text string, "Uninitialized," and are automatically disabled.

If you set a combo box as a string data type (such as string or character), when runtime initializes the control and loads the -- Select One -- prompt into the combo box, the prompt's internal key value is "" (an empty string). If you set a combo box as a numeric data type (such as Integer, ID, or Math Numeric), when runtime initializes the control and loads the -- Select One -- prompt into the combo box, the prompt's internal key

value is -999999999. It is therefore important, when you are building your application ERs, to allow for the case when the user leaves the control set selected on the default prompt item and the key value is sent from the control.

The control initialization process is illustrated in this flowchart:

Figure 19–1 Combo box control initialization process



Generally two cases exist where runtime might show an item description other than the -- Select One -- prompt as the currently selected item in a combo box when a form appears initially:

- When a given combo box's associated DD item has a default value defined and you come into the form in Add mode, runtime attempts to match the default value to the keys of the items currently loaded in the combo box and sets that item as the selected item.

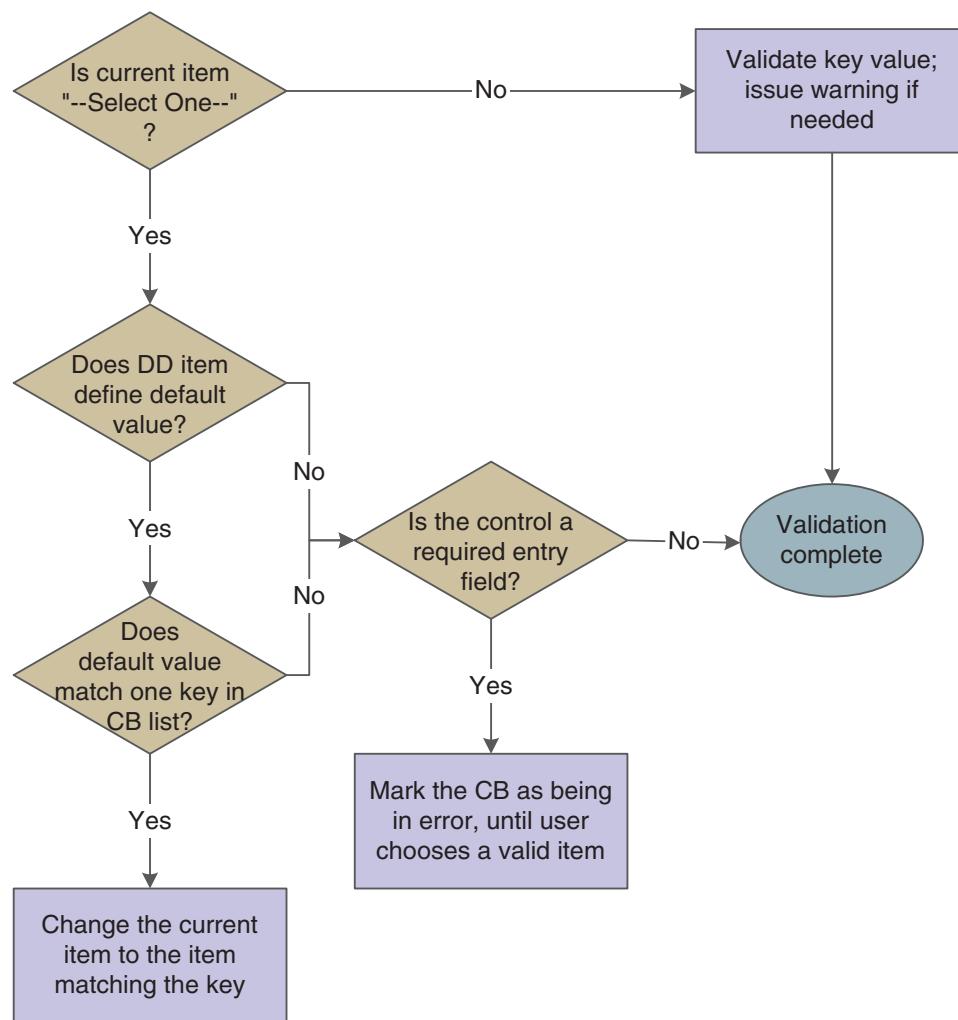
- Dependent on form type, regardless of whether the form was started using form interconnect, the current form mode (Add/Copy/Update), and whether the control is mapped to a business view column (Key or otherwise), runtime might have retrieved a key value also and will try to set this as the current selected item in the combo box.

If you want to load the combo box from cache or using the **Add Item** system function, the event most typically used for such loads is **Post Dialog Initialized**.

19.5.2 Control Validation

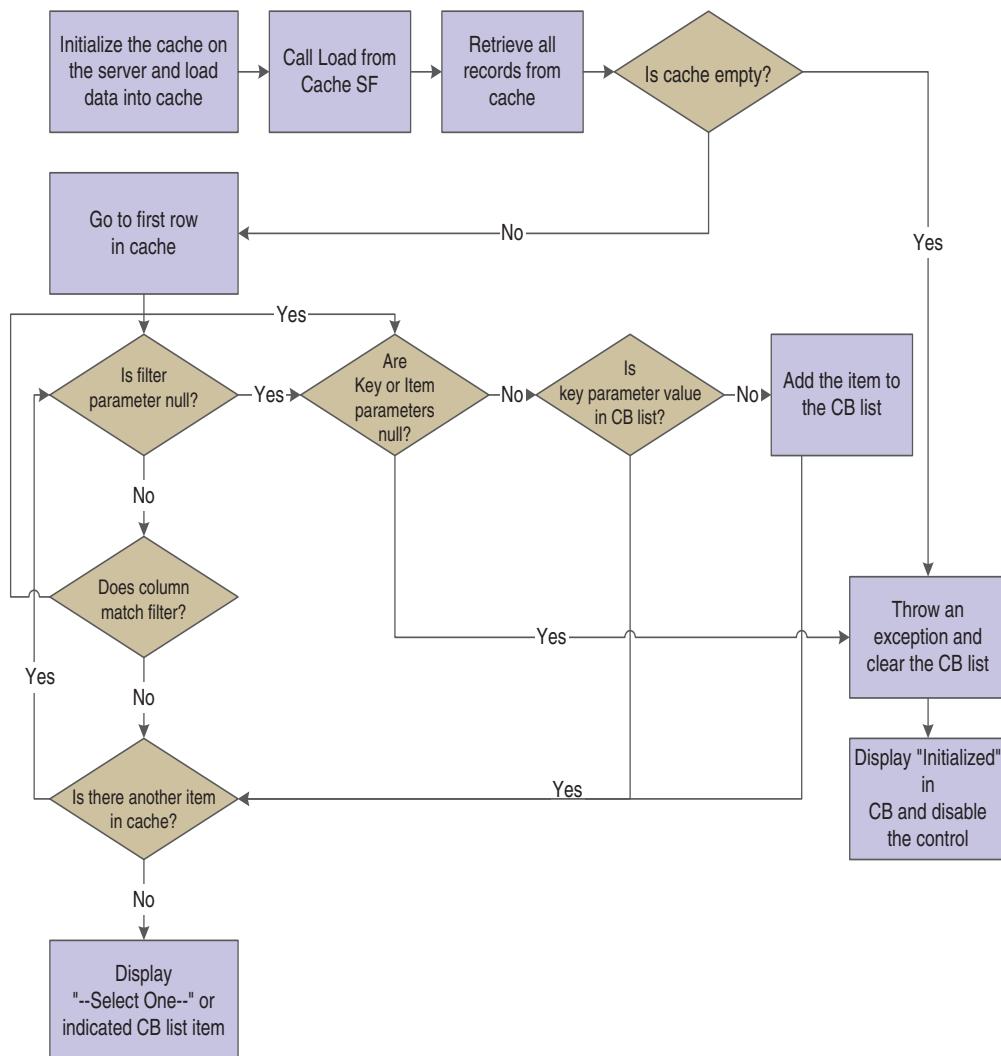
Runtime performs the validation process illustrated in this flowchart when the control is validated:

Figure 19–2 *Combo box control validation process*



19.5.3 Load from Cache

This flowchart illustrates how runtime loads the combo box from cache when the **Load from Cache** system function is processed. If an error occurs at any point during the processing, any data that might have been loaded into the combo box is cleared and the control is disabled:

Figure 19–3 *Combo box control load from cache process*

19.5.4 Database Commit

When an application signals a database commit, runtime writes the key value (instead of the displayed description) of the currently-selected combo box item to the database. If the field is flagged as not required and the currently-selected value is -- Select One --, runtime saves a string of continuous blank characters matching the size of the associated table column. The system cannot save the actual matching empty string ("") key value associated with the -- Select One -- prompt to the database, which is why it saves a string of blank characters instead.

19.5.5 System Functions

When you call **Set Data Dictionary Item**, **Set Data Dictionary Override**, or **Load From Cache**, runtime clears the control of its current items prior to loading any new data specified by the system function call. This is not only true of combo boxes on a form, but in a grid as well. If the new data dictionary item associated with the system function call or associated UDC override results in a change from the current DD Item and/or UDC specified for the grid control, then runtime removes all list items from all embedded combo box controls for all current rows of the column, except for the

default -- Select One -- prompt. If this action results in a new set of UDC values, then runtime loads those values into every current embedded combo box control for all of the rows in the grid or Parent Child control.

19.5.6 Import into Grid

HTML users can import data into a grid containing embedded combo boxes just as they can into a grid without the JD Edwards EnterpriseOne grid import/export feature. When a user does so, the value being imported into the embedded combo box must be the Key parameter value that corresponds to a valid combo box key/item (description) pair.

If the target cell resides in a grid column that has a data dictionary item with an associated UDC, then runtime creates an embedded combo box in the target cell and loads the UDC-defined key/item pairs into it. Next, runtime attempts to match the imported key to one of the currently loaded keys. If it finds a match, runtime sets it as the currently-selected item in the cell. Otherwise, it ignores the item and sets the cell to the -- Select One -- value.

If the target cell resides in a grid column that does not have a data dictionary item with an associated UDC, then runtime creates an embedded combo box with one item set as selected having a Key parameter of (blank) and an Item (description) parameter of Uninitialized. Then runtime disables the cell.

19.6 Combo Box Control System Functions

This section discusses the system functions unique to the combo box control when it resides on a form (that is, it is not embedded in a grid).

Add Item

Use this system function to add an item to the combo box if it is not displaying a list that is based on a UDC.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Key

Input, required. The key of the item to add (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Item

Input, required. The name of the item to add (string). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Index

Input, optional. The zero-based index position where the item is to be added (integer). If you do not specify this parameter, runtime appends the item to the bottom of the list. The default -- Select One -- prompt always occupies index position zero. If an invalid index is specified, the addition of the new item fails and an error message is logged. Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Additional Notes

Use this system function to add an item to the combo box. Runtime verifies that the Key and Item parameter values are both specified. If either is missing, the addition of the new item fails and an error is logged. If you try to insert an item having a key value that already exists in the list, runtime will not add the item and will log an error. The combo box control must reside on the form (as opposed to being embedded in a grid). The control must have a valid data dictionary item associated with it.

If the key value of the key/description pair specified in the **Add Item** system function call to be added to the control corresponds to a valid key value in the associated UDC set and that key value is currently not loaded in the control (in other words, it was removed after the control was initialized), the runtime engine allows the item to be added back into the control. Runtime recognizes the fact that it currently does not exist in the control's item list and that it is a valid key value in the UDC's set. Otherwise, the attempt to add the new item will fail.

Get Description

Use this system function to get the displayed description of the current selected item of a specified combo box.

Parameters

ComboBox

Input, required. The combo box FC to affect.

Item

Input, required. The object to which to assign the return value that designates the description text of the current selected item (string). Set the parameter to an applicable object from the object list.

Returns

This system function returns the name of a combo box item to the object identified by Item.

Get Index of Key

Use this system function to get the zero-based index position of an item in a combo box list by specifying its key. If the key value specified does not exist in the combo box's current set of items, no index value is returned and an error message is logged.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Key

Input, required. The key of the target item (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Index

Input, required. The object to which to assign the return value that designates the zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Returns

This system function returns the zero-based index position of a combo box item to the object identified by *Index*.

Get Item at Index

Use this system function to get the displayed description of an item in a combo box list by specifying its index position in the list. If the index value specified is invalid, no description is returned and an error message is logged.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Index

Input, required. The zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Item Description

Input, required. The object to which to assign the return value that designates the displayed description text of the target item (string). Set the parameter to an applicable object from the object list.

Returns

This system function returns the name of a combo box item to the object defined by Item Description.

Get Item Count

Use this system function to determine the total number of items in the combo box list including the -- Select One -- prompt.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Number

Input, required. The object to which to assign the return value that designates the number of list items. Set the parameter to an applicable object from the object list.

Returns

This system function returns the number of items in the combo box list to the object identified by Number.

Get Key at Index

Use this system function to get the key of an item in a combo box list by specifying its index position in the list. If the specified index value is invalid, no key value is returned and an error message is logged.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Index

Input, required. The zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Key

Input, required. The object to which to assign the return value that designates the key of the target item (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Returns

This system function returns the key of a combo box item to the object defined by Key.

Load from Cache

Use this system function to load the combo box control from a JD Edwards EnterpriseOne cache. The control must be based on a DD item which does not have an associated UDC defined for it.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Cache Name

Input, required. The name of the prepopulated JD Edwards EnterpriseOne cache from which to load combo box items (string). Set the parameter to an applicable object from the object list.

Data Structure

Input, required. The name of the predefined JD Edwards EnterpriseOne data structure to use to load the combo box items from cache (string). This must be the same data structure as the one used to initialize and load the cache. Set the parameter to an applicable object from the object list.

Key Column

Input, required. The cache column containing the key values for the items to be loaded into the combo box (type defined by data structure). Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Item Description Column

Input, required. The cache column containing the text descriptions for the items to be loaded into the combo box (type defined by data structure). Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Filter Column

Input, optional. The cache column containing the values against which to filter items to be loaded into the combo box (type defined by data structure). The data type must match that of the Key Column parameter. Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Filter Value

Input, optional. A value used by runtime to compare against values specified in the Filter Column parameter (string or numeric). The data type must match that of the Filter Column parameter itself. After it retrieves all of the cache records, runtime loads a key/description pair into the combo box only if its Filter Value matches the Filter Column value for that record. Otherwise the retrieved record is ignored and runtime proceeds to process the next retrieved record. Set the parameter to <N/A> or an applicable object from the object list.

Additional Notes

The **Load From Cache** mechanism always retrieves all of the records currently loaded in the cache from the enterprise server to the Web server when the call is made regardless of whether the optional Filter Column and Filter Value parameters are specified. If you include several combo boxes on a form, you might decide to create a single data structure that loads the cache for all of them. However, you must use

discretion in balancing how many records are loaded simultaneously in the cache versus the performance penalties that might result when loading a single combo box with a relatively small subset of the cache.

Remove Item by Index

Use this system function to remove an item from a combo box list by specifying its index position.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Index

Input, required. The zero-based index position of the item to remove (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Optional

Input, required. A special value indicating whether to remove all items. Set the parameter to <Remove All> or <N/A>.

Additional Notes

You can also clear all items from the list. If the index value specified is invalid and the Optional parameter equals <N/A>, no item is removed from the control and an error message is logged.

Remove Item by Key

Use this system function to remove an item from a combo box list by specifying the item's key.

Parameters

ComboBox Control

Input, required. The combo box FC to affect.

Key

Input, required. The key of the item to remove (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Optional

Input, required. A flag indicating whether to remove all items. Set the parameter to <Remove All> or <N/A>.

Additional Notes

You can also clear all items from the list. If the key value specified is invalid and the Optional parameter equals <N/A>, no item is removed from the control and an error message is logged.

Select Item

Use this system function to set one of the items in the combo box as the currently selected item programmatically. The item's description appears in the control just as if the user had chosen it.

Parameter

ComboBox Control

Input, required. The combo box FC to affect.

Key

Input, required. The key of the item to set (string or numeric). Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Additional Notes

If the key value specified is invalid, the combo box's selection remains unchanged.

Embedded Combo Box System Functions

This section discusses the system functions unique to the combo box control when it is embedded in a grid.

Add Item

Use this system function to add an item to the combo box list embedded in a grid that is not displaying a list based on a UDC.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Key

Input, required. The key of the item to add (string or numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Item

Input, required. The name of the item to add (string). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Index

Input, optional. The zero-based index position where the item is to be added (integer). If you do not specify this parameter, runtime appends the item to the bottom of the list. The default -- Select One -- prompt always occupies index position zero. If an invalid index is specified, the addition of the new item fails and an error message is logged. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Additional Notes

Use this system function to add an item to the combo box if it is not displaying a list that is based on a UDC (however, the control itself must be based on a valid DD item). Runtime verifies that the Key and Item parameter values are both specified. If either is missing, the addition of the new item fails and an error is logged. If you try to insert an item having a key value that already exists in the list, runtime will not add the item and will log an error.

Get Description

Use this system function to get the name of an item in a combo box list by specifying its row number.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Item

Input, required. The object to which to assign the return value that designates the description text of the current selected item (string). Set the parameter to an applicable object from the object list.

Returns

This system function returns the name of a combo box item to the object identified by Item.

Get Index of Key

Use this system function to get the zero-based index position of an item in a combo box list by specifying its key. If the key value specified does not exist in the combo box's current set of items, no index value is returned and an error message is logged.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Key

Input, required. The key of the target item (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Index

Input, required. The object to which to assign the return value that designates the zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Returns

This system function returns the zero-based index position of a combo box item to the object identified by Index.

Get Item at Index

Use this system function to get the displayed description of an item in a combo box list by specifying its index position in the list. If the index value specified is invalid, no description is returned and an error message is logged.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Index

Input, required. The zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Item Description

Input, required. The object to which to assign the return value that designates the displayed description text of the target item (string). Set the parameter to an applicable object from the object list.

Returns

This system function returns the name of a combo box item to the object defined by Item Description.

Get Item Count

Use this system function to determine the total number of items in the combo box list including the -- Select One -- prompt.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Number

Input, required. The object to which to assign the return value that designates the number of list items. Set the parameter to an applicable object from the object list.

Returns

This system function returns the number of items in the combo box list to the object identified by Number.

Get Key at Index

Use this system function to get the key of an item in a combo box list by specifying its index position in the list. If the specified index value is invalid, no key value is returned and an error message is logged.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Index

Input, required. The zero-based index position of the target item (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Key

Input, required. The object to which to assign the return value that designates the key of the target item (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Returns

This system function returns the key of a combo box item to the object defined by Key.

Load from Cache

Use this system function to load the combo box control from a JD Edwards EnterpriseOne cache. The control must be based on a DD item which does not have an associated UDC defined for it.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Cache Name

Input, required. The name of the prepopulated JD Edwards EnterpriseOne cache from which to load combo box items (string). Set the parameter to an applicable object from the object list.

Data Structure

Input, required. The name of the predefined JD Edwards EnterpriseOne data structure to use to load the combo box items from cache (string). This must be the same data structure as the one used to initialize and load the cache. Set the parameter to an applicable object from the object list.

Key Column

Input, required. The cache column containing the key values for the items to be loaded into the combo box (type defined by data structure). Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Item Description Column

Input, required. The cache column containing the text descriptions for the items to be loaded into the combo box (type defined by data structure). Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Filter Column

Input, optional. The cache column containing the values against which to filter items to be loaded into the combo box (type defined by data structure). The data type must match that of the Key Column parameter. Set the parameter to an applicable object from the object list (the list is empty until you select a data structure).

Filter Value

Input, optional. A value used by runtime to compare against values specified in the Filter Column parameter (string or numeric). The data type must match that of the Filter Column parameter itself. After it retrieves all of the cache records, runtime loads a key/description pair into the combo box only if its Filter Value matches the Filter Column value for that record. Otherwise the retrieved record is ignored and runtime proceeds to process the next retrieved record. Set the parameter to <N/A> or an applicable object from the object list.

Additional Notes

The **Load From Cache** mechanism always retrieves all of the records currently loaded in the cache from the enterprise server to the Web server when the call is made regardless of whether the optional Filter Column and Filter Value parameters are specified. If you include several combo boxes on a form, you might decide to create a single data structure that loads the cache for all of them. However, you must use discretion in balancing how many records are loaded simultaneously in the cache versus the performance penalties that might result when loading a single combo box with a relatively small subset of the cache.

Remove Item by Index

Use this system function to remove an item from a combo box list by specifying its index position.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Index

Input, required. The zero-based index position of the item to remove (integer). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Optional

Input, required. A special value indicating whether to remove all items. Set the parameter to <Remove All> or <N/A>.

Additional Notes

You can also clear all items from the list. If the index value specified is invalid and the Optional parameter equals <N/A>, no item is removed from the control and an error message is logged.

Remove Item by Key

Use this system function to remove an item from a combo box list by specifying the item's key.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Key

Input, required. The key of the item to remove (string or math-numeric). Set the parameter to an alphanumeric constant (<Literal>) or applicable object from the object list.

Optional

Input, required. A flag indicating whether to remove all items. Set the parameter to <Remove All> or <N/A>.

Additional Notes

You can also clear all items from the list. If the key value specified is invalid and the Optional parameter equals <N/A>, no item is removed from the control and an error message is logged.

Select Item

Use this system function to set one of the items in the combo box as the currently selected item programmatically. The item's description appears in the control just as if the user had chosen it.

Parameters

Grid Control

Input, required. The grid FC to affect.

Grid Column

Input, required. The grid column to affect. Set the parameter to an applicable object from the object list.

Grid Row

Input, required. The grid row to affect (integer). Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or an applicable object from the object list.

Key

Input, required. The key of the item to set (string or numeric). Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Additional Notes

If the key value specified is invalid, the combo box's selection remains unchanged.

20

Understanding Edit Controls

This chapter contains the following topics:

- [Section 20.1, "Edit Controls"](#)
- [Section 20.2, "Edit Control Events"](#)
- [Section 20.3, "Edit Control Runtime Processing"](#)
- [Section 20.4, "Edit Control System Functions"](#)

20.1 Edit Controls

An edit control is a text field on a form. All form types can contain edit controls except message forms. Two types of edit controls are available. The first type is commonly referred to as a database field. It is associated with an item in the business view (BV) and through that connection to a specific data dictionary (DD) item. Database fields represent a field in a database record. The second type of edit control is commonly referred to as a DD field, and it also has a connection to a specific DD item.

Within the realm of database fields an additional distinction between filter fields and nonfilter fields exists. Database fields are nonfilter fields by default. Filter fields are used to alter the selection criteria of a database fetch. A filter can have a comparison type of equal, not equal, less than, less than or equal to, greater than, or greater than or equal to. A filter can be marked such that a wildcard (*) displays when the filter is not included in the selection.

The storage for the value of the edit control is based on the type of its associated DD item (such as numeric, string, character). An edit control is affected by the properties of the associated DD item. For example, if an edit control is associated with a DD item of type string with a length of thirty, the edit control will not permit more than thirty characters to be typed into the field.

Edit controls are generic input fields and have no associated text. You must associate edit controls with data items.

If you associate an edit control with a data item from the BV associated with the form, then the value entered by a user at runtime updates the table. If you associate an edit control with a data item not associated with the BV, then the value entered at runtime is for display only. However, you can make a field read-only for BV data items by enabling the Read Only property. You can also associate an edit control with a DD item with a user-defined code (UDC).

Edit controls have a Type Ahead feature. When a user enters a character in the field, the system searches a history list for a match. If a match exists, it appears in the field and is highlighted. This feature is particularly useful for data entry work because it can reduce the amount of typing required. A user can enable or disable Type Ahead

editing in JD Edwards EnterpriseOne. Additionally, the Type Ahead capabilities of the browser (such as Internet Explorer) must be enabled as well. Type Ahead is disabled for Chinese, Japanese, and Korean (CJK) languages and multiline edit controls.

As you add controls to the form, you can indicate how the runtime engine filters the incoming records from the database. For example, if the find/browse form has two controls on which you want to filter, the resulting SQL statement that you generate will have an AND condition for each condition. For example, if you have Search Type and Alpha Description as the controls on the form, the filter criteria for Search Description should be >= and the Search Type control should be =. If a user types D and puts a V in the Search Type field, this is the resulting SQL statement:

```
SELECT * FROM F0101 WHERE (ABAT1 = V AND ABDC LIKE D%) ORDER BY ABAN8 ASC
```

You might also want to apply filters to edit controls when records need to fall between two values. In this case, you use a range filter. For example, in distribution, a status is assigned to each line of the order. One status is the current status and the other status is the next status. In this example, you filter records that are greater than or equal to the present status and less than or equal to the next status. You drop one value, filter the value, and then drop the next value.

A consideration when using the Required Field flag with edit fields is that, if the required field indicator should be displayed in the application, you must check the Required Field property for the edit field and then connect static text to it. Be sure to size the connected static text box appropriately to allow for the "required" indicator to be appended to the end of the text, and also to allow for translations. If you do not connect static text to the edit field, runtime does not display the required field indicator in the application.

To connect static text to the edit field, select the edit field and the label. Then select Edit, Reconnect. To disconnect the static text from the edit field, select them and then select Edit, Disconnect.

20.2 Edit Control Events

These events can fire on an edit control:

- **Control is Entered**
- **Visual Assist Button Clicked**
- **Post Visual Assistant Clicked**
- **Control is Exited**
- **Control is Exited/Changed-Inline**
- **Control is Exited/Changed-Asynch**

20.3 Edit Control Runtime Processing

This section discusses how the runtime engine processes the edit control.

20.3.1 Control is Entered

The control can acquire focus either through a user action, such as by pressing the Tab key or clicking it with the mouse, or through a system command. When the control acquires focus, runtime fires the **Control is Entered** event.

20.3.2 Control is Exited

When the user leaves the control (such as by pressing the Tab key), the system validates the control and saves the value to memory. A number of events are fired during this process.

Data validation consists of ensuring that the value meets any established criteria for the DD item upon which it is based, including size, type, and so forth. Additionally, runtime compares the current value of the control with the value of the control before it started the exit process. If the value has changed, then it fires two additional events.

20.4 Edit Control System Functions

This section discusses system functions for edit controls, which reside in the Control folder.

Set Edit Control Color

Use this system function to set the background color for the edit control.

Parameters

Control

Input, required. The FC to affect.

Color

Input, required. The color to which to set the control. Double-click <Pick Color> to select a specific color. Otherwise, set the parameter to <Pick Color> or the default color (<Reset Color>).

Set Edit Control Font

Use this system function to set the font and font characteristics (for example, typeface and color) for an edit control.

Parameters

Control

Input, required. The FC to affect.

Font

Input, required. The font and font characteristics to apply. Double-click **<Pick Font>** to select specific settings. Otherwise, set the parameter to **<Pick Font>** or the system default settings (**<Reset Font>**).

21

Understanding Grid Controls

This chapter contains the following topics:

- [Section 21.1, "Grid Controls"](#)
- [Section 21.2, "Grid Control Design-Time Considerations"](#)
- [Section 21.3, "Grid Control Events"](#)
- [Section 21.4, "Grid Control Runtime Processing"](#)
- [Section 21.5, "Grid Control System Functions"](#)

21.1 Grid Controls

A grid control is similar to a spreadsheet. Use grids to display data and to enable users to enter information. Unlike an edit control, grid controls can show multiple data items and multiple table rows at once. You also can use grid controls to enable users to edit table records. In end-user documentation, grid controls are referred to as detail areas.

A grid can either be a browse grid or an update grid. You can use a browse grid for viewing only, and you cannot select individual cells. The find/browse, search & select, power browse, and portlet browse forms have browse grids, as do browse subforms.

You can use an update grid to add, delete, or update records. Cells in an update grid can be selected individually. The header detail, headerless detail, power edit, and portlet edit forms have update grids, as do update subforms.

Grid controls of both types contain columns. The columns are specified at design time and are one of these types:

- Database column or
- Data dictionary (DD) column.

A database column is associated with an item in the business view (BV) and through that connection to a DD item. Database columns represent a field in a database record.

Although only one type of column is referred to as a DD column, both types have a connection to a specific DD item. The difference is that a database column has the additional connection to a BV field. A grid column is affected by the properties of the associated dictionary item. For example, if a grid column is associated with a dictionary item of type string with a length of 30, that grid column will not permit more than 30 characters to be typed into the cell.

Grid controls can also have a query-by-example (QBE) line. The QBE columns have a one-to-one correspondence with the grid columns. You use a QBE value to change the selection criteria of a database fetch. Only database columns enable entry in the QBE

columns because the purpose of the QBE is to affect the selection and only database columns are in the BV. A QBE column can have one of these comparison types:

- Equal.
- Not equal.
- Less than.
- Less than or equal to.
- Greater than.
- Greater than or equal to.

The comparison type is equal unless you specify otherwise. You can specify the comparison type in the QBE column or by using system functions. You can use wildcards (*) or (%) for an inexact search on a string field.

Grid values can be retrieved on a cell-by-cell basis or on a row-by-row basis. Internal storage for the grid columns exists in the runtime engine. The way the grid column value is stored is based on the type of the associated dictionary item (for example, math numeric, string, character), and it is distinct from the screen representation of the value. Only one row of data can be acted upon at any given time. Each event executes in the context of a specific row.

21.2 Grid Control Design-Time Considerations

Grid columns can have a heading row and might also have a QBE row to facilitate user searches. Additionally, you can specify rows or columns to display information that you acquire or calculate during runtime. Typically, the rest of the rows contain data. Generally, this data comes from database tables.

Usually, when you make the grid control, you place data items in the grid. The data items can come from the BV attached to the form, or they can be based on DD items. Either way, each data item becomes a column. Just as the grid control itself has a variety of property settings, you can set property values for each column in the grid as well.

You have a great deal of control, both at design time and at runtime, over how the grid appears, what it displays, and how it functions.

This section discusses how to:

- Design the grid.
- Add columns to the grid control.
- Display grid data as icons.
- Set property values for the grid control.
- Show multiple currencies per column.

21.2.1 Designing the Grid

This task outlines the overall process for creating a grid.

To design the grid:

1. Create a form.

Find/Browse, header detail, headerless detail, and search & select forms have a grid control on them automatically.

Power edit, power browse, reusable and embedded edit subforms, reusable and embedded browse subforms, edit portlet, and browse portlet forms will accept a grid control.

2. On the form level, attach a BV for the engine to use to populate the grid or use values assigned in ER and system functions to insert grid rows.

3. Drop a grid control onto the form.

4. Configure the control.

If you are working with a header detail form, you can attach a different BV to the control.

5. Add columns to the control.

6. Configure each column.

7. Add logic.

21.2.2 Adding Columns to the Grid Control

To add columns to a grid control:

1. Perform one of these tasks:

- To add a data item to the grid from the BV associated with the form, double-click a data item in the Business View Columns Browser.
- To add a data item to the grid that is not in the BV associated with the form, search for a data item in the Data Dictionary Browser, and then double-click it.

Each time you double-click an item, Form Design Aid (FDA) adds it as a column to the grid control.

2. Repeat step one until you have added all of the columns you need.

3. Double-click the grid control.

4. On Grid Properties, click the Columns tab and arrange the column order by selecting a column and then using the Up and Down buttons to shift its position in the list.

5. Click the Sort Order tab and set the order in which the system will sort returned records by performing these steps:

- a. Select a data item in the Unsorted Columns list and click the right arrow. Repeat for each data item on which you want to sort.

The data item moves to the Sorted Columns list.

- b. To toggle between displaying by ascending and descending order, click the data item in the Sorted Columns list.

The letter A next to the data item indicates that the system will display records in ascending order. The letter D next to the data item indicates that the system will display records in descending order.

- c. Arrange the column sort order by choosing a column in the Sorted Columns list and then using the Up and Down buttons to shift its position in the list.

21.2.3 Displaying Grid Data as Icons

You can display grid cell data as icons to provide quicker, more meaningful visual feedback on certain types of data, such as status flags or other data where the internal

value of a flag is not directly meaningful to a user. The ability to display grid data as icons enables you to associate values or ranges of values with an icon image, and then display that image in a grid, instead of the underlying data value.

You can optionally make icons clickable so that a behavior is associated with them. You can also optionally create tooltip popups to display supplementary information about the data. These tooltips are displayed when the user hovers the mouse pointer over the icon image.

21.2.3.1 Icon Display States

An icon can have several states. It can be clickable or non-clickable, and it can be enabled or disabled. It can also display a slightly different image when the mouse hovers over it. The appearance of the icon should be different in each case. This enables the user to understand quickly the current interactivity state of a grid cell.

Icons are therefore represented physically by a group of related graphic images, each of which corresponds to one of the different icon states. The naming convention enables these graphic images to be an interrelated set grouped under a single icon. The base or normal image represents the icon under normal, non-clickable conditions. Standard suffixes identify the other, related images for that same icon.

This table shows the icon naming convention:

State	Filename Suffix	Example
Normal (non-clickable)	None	status_ok.gif
Clickable	_cl	status_ok_cl.gif
Clickable – mouseover	_mo	status_ok_mo.gif
Clickable – disabled	_di	status_ok_di.gif

The graphic image types can be in gif, bmp, jpg and png formats. For each new icon you want to use, you must create four images (one for each state), and then store those images in <E1_install_dir>\system\JAS\EA_JAS_80.ear\webclient.war\img directory.

21.2.3.2 Associating Icons to Data Values

To use an icon to represent an underlying data value or range of values in a column, you use a client-side NER. Client-side NER offers some key advantages when representing data values as icons:

- *Reusability.* The NER can be used from any icon column. An example is that you use a NER to capture the decision process associated with each icon data item and then use the same NER in all columns displaying that same data item.
- *Consistency.* The same type of data is represented with the same set of icons on multiple grids, over multiple applications.
- *Flexibility.* When you need a one-off or non-standard icon representation, you can create special NERs. Alternately, you can use data structures to pass more information than the current value of that single data column to the icon decision-making process. This enables more flexible icon representation.

21.2.3.3 Tooltips on Icons

Displaying data as icons has the disadvantage of hiding detailed and possibly necessary information about the icon from the user.

Tooltips enable you to augment the icon with detailed information. When the user's mouse pointer hovers over an icon, the extended information about that grid cell is displayed. The information that displays is specific to the form and the grid column, but can include any or all of:

- Icon-specific text for the currently displayed icon.

This text can, for example, provide short, explanatory context information to the user about why the icon is displayed, or its meaning when it is selected for display. This tooltip is labeled Show Icon Tooltip in the dialog box.

- DD Name for the column.

This tooltip is labeled Show DD Name Tooltip in the dialog box.

- The underlying data value.

This tooltip is labeled Show Value Tooltip in the dialog box.

- DD Alias for the column.

This tooltip is labeled Show DD Alias Tooltip in the dialog box.

21.2.3.4 Implementing Icons on Grids

Use these steps to display grid data as icons:

1. Identify data values or ranges of values that you want to represent as icons, and then decide which icons to use to represent them. Ensure that all required images for all icons are in the designated location.
2. Create the icon-setting (client-only) NER to define the value-to-icon relationships:
 - a. Define a data structure that enables enough information to be passed to the NER to make all icon-setting decisions.
 - b. Use ER to define all necessary data value conditions.
 - c. Provide at least one call to the **Set Grid Cell Icon** system function. Each call to this system function enables you to select the icon's base (normal) image using the Image Picker and then to optionally set icon-specific tooltip text to apply under that specific data
3. Associate the NER function that sets the icon with the grid columns:
 - a. From the Column Properties dialog, set Display Style to **Icon**.
Update the ER for the **Grid Cell Display Changed** event by adding a call to the NER function that sets the icon.
 - b. Optionally, to make an icon clickable, select Clickable, and then update the ER for the **Grid Column Clicked** event

21.2.4 Setting Property Values for the Grid Control

Use design-time settings in FDA to affect grid display (including showing multiple currencies per column), loading and processing, and how a user can enter data.

In the next subsections, it is assumed that you access all property values from the Property Browser in FDA.

21.2.4.1 Grid Control Display

You control grid appearance through a combination of property values that you set at design time and system function values you program to occur during runtime.

Consider carefully the design-time settings you make because you may not be able to change them during runtime.

Desired Result	To Implement	Notes
Enable users to click a column value.	Select the column and set Clickable to Yes.	Runtime fires the Grid Column Clicked event when the user selects the column value.
Change the appearance of the grid based on HTML code.	Select the grid control, set Use Alternate Grid Format to Yes .	N/A
Display icons in grid cells.	Select a column and set Display Style to Icon .	Runtime fires the Grid Cell Display Changed event whenever a column value is set for an icon column. You must provide a call to a NER that makes at least one call to the Set Grid Cell Icon system function, or the cell will not display an icon.
Add tooltips to icons.	Select a column, set Display Style to Icon , and choose any number of: <ul style="list-style-type: none"> ▪ Show Icon Tooltip ▪ Show Value Tooltip ▪ Show DD Name Tooltip ▪ Show DD Alias Tooltip 	Each selected component of the tooltip, if any, is displayed on a separate line in the order listed.
Disable the QBE row.	Select a column and set Disable QBE to Yes .	You can only disable the QBE on a column-by-column basis. Alternatively, you can hide the entire row with the grid property, Hide Query By Example.
Display multiple currency types in a single column.	Select a column and set Support Multiple Currencies to Yes .	
Hide a column if currency is disabled.	Select a column and set No display if currency is OFF to Yes .	
Hide the row selector cell at the beginning of each row (check box or radio button) which shows the rows that have been selected.	Select the grid control and set Hide HTML Row Selector to Yes .	
Hide the Query-By-Example (QBE) row.	Select the grid control and set Hide Query By Example to Yes .	If you hide the QBE row, not only do you prevent user interaction, but you also prevent program interaction. You cannot use system functions to work with a hidden QBE row. Alternatively, you can disable QBE cells on a column-by-column basis with the column property, Disable QBE.

Desired Result	To Implement	Notes
Hide the row headers.	Select the grid control and set Hide Row Numbers to Yes .	
Open the form in update mode.	Select the grid control and set Update Mode to Yes .	
Override the grid column header.	Select the column and enter the text in Column Header One and Column Header Two that you want to appear.	Column Header One is the first line and Column Header Two is the second.
Prevent users from customizing the grid.	Select the grid control and set Display Customize Grid to No .	When disabled, the system will display neither the "Customize Grid" link on the grid header nor the "Grid Formats" menu item.
Reorder the columns in the grid.	Select the grid control, select Column Order, and launch the Grid Properties form. Select the column to move and then click the Up or Down buttons to change its location relative to the other columns.	
Set the number of rows to display at a time.	Select the grid control and set Grid Row Count to the number of rows you want to display at a time.	The JDE.ini property, GlobalPageSize also controls the number of rows to display at a time. Runtime uses whichever value is greater.
Wrap text in the column	Select the column and set Wrap Text to Yes .	When disabled, the system truncates text that exceeds the column width.

In addition to these property settings, use these system functions to affect grid appearance:

- **Set Grid Color**
- **Set Grid Font**
- **Set Grid Row Format**

21.2.4.2 Loading and Processing Behavior

FDA provides some options to automate fetches. You can choose to load the grid based on the fetch logic when the user launches an application if the grid resides on a find/browse or headerless detail form. Use automatic loading options with care; in some cases, enabling these features can have a significantly negative impact on performance.

Especially in HTML environments, the page-at-a-time processing feature can enhance performance. This option enables you to populate the grid at the rate of one page at a time (the runtime engine only loads enough data to fill the grid, and then loads enough data to fill the grid again if the user clicks Next). Page-at-a-time processing is enabled by default, and the JD Edwards standard is to leave it enabled for all form types.

Desired Result	Navigation	Notes
Enable users to select multiple rows.	Select the grid control and set Multiple Select to Yes .	
Change the sort order.	Select the grid control, select Column Sort Order, and launch the Grid Properties form.	
	Move columns to the right that you do want to sort on.	
Load a different data item.	Select a column, select Data Item Information, and launch the Grid Column Properties form. Select the Data Items tab and select the data item to use.	
Load data at the rate of one grid page at a time (page-at-a-time).	Select the grid control and set Disable Page-at-a-Time Process to No .	
Load data automatically upon changes in a related (child) form.	Select the grid control and set Auto Find On Changes to Yes .	Use this option only on forms that have no modeless form interconnections.
Prevent users from importing or exporting files.	Select the grid control and set one or more of these properties to No : <ul style="list-style-type: none">▪ Display Import from Excel▪ Display Export to Excel▪ Display Export to Word	If a property is disabled, the system will not display the related option icon or menu item.
Process all the grid rows on a database commit, not just the ones that changed.	Select the grid control and set Process All Rows in Grid to Yes .	When enabled, this property causes runtime to apply row changed and row exited logic to all rows, no matter their state.

21.2.4.3 Data Entry Behavior

These properties affect how the user interacts with the grid.

Desired Result	Navigation	Notes
Post rows silently to improve application performance if you are running in low interactivity mode.	Select the grid control and set Multi-Line edit to Yes .	When enabled, runtime posts rows in groups of three to five to the database in the background. When disabled, in low interactivity mode, runtime posts each time the user tabs out of the row, forcing the user to wait for a refresh before continuing.
Enable users to sort on a column.	Select the column and set Sortable by end user to Yes .	When enabled, users can choose to sort grid rows in ascending or descending order based on this column.

Desired Result	Navigation	Notes
Make a column a check box column.	Select the column and set Display Style-Default or Checkbox or Combobox to Checkbox .	The Column Selection Changed event fires when the user selects or clears a check box in a check box column.
Make a column a combo box column.	Select the column and set Display Style-Default or Checkbox or Combobox to Combobox .	The Column Selection Changed event fires when the selected item in the combo box control is changed through user input, or programmatically. The Column is Exited event fires when the user exits the column.
Prevent user from adding lines to a grid.	Select the grid control and set No Adds on Update Grid to Yes .	Although they cannot add new records, users can still edit existing ones.
Prevent users from entering data, or making a selection.	Select the column and set Disabled to Yes .	
Require the user to input a value or make a selection in a column.	Select the column and set Required Field to Yes .	When set, runtime processes the Required Field property to display the required field indicator (an asterisk), next to the grid column header label. Set the grid column sufficiently wide to allow room for the required indicator character at the end of the column header text, and also wide enough to facilitate translation.

21.2.5 Showing Multiple Currencies per Column

When a column has the Support Multiple Currencies option enabled, the runtime engine assumes that each cell contains its own currency setting, and it formats each cell based on that cell's currency decimal setting. The runtime engine will not apply the currency settings to other grid rows, however. Therefore, the application needs to apply currency to each grid row individually. For example, the Amount column in row 1 might have a JPY currency type and be formatted with no decimals, while the Amount column in row 2 might have a USD currency type and be formatted with two decimals.

When a column has the Support Multiple Currencies option disabled, the runtime engine assumes that all of the cells in that column share the same currency setting, and so it applies that currency setting to other grid rows. Therefore, if you specify the currency setting in one row, the system overwrites the currency setting for all the other rows in the grid to match. This feature offers a performance benefit for those grids that contain only one currency because the application needs to specify a currency setting to one grid row only to affect the entire grid.

These currency rules apply:

- When assigning values using conventions such as target = source, if the source object does not have any currency information (currency code = null or empty string), then the target object keeps its own currency.
- When a GB object is cleared, the currency code and currency decimal information for that column is not cleared.

21.2.6 Adding Aggregation to Grid Column

To add aggregation to a grid column:

1. Access the grid column properties.
2. Select Support Aggregation check box.
3. Access event rules for the grid column.
4. Add event rules on the Aggregation Clicked event.
5. Override the visual assist for the grid column.

21.3 Grid Control Events

This section discusses the events that runtime fires while processing a grid control.

Depending on the type and mode of the form, runtime fires a variety of events in response to events regarding the grid control.

If you set up a custom fetch routine, you will want to make use of **Get Custom Grid Row**. To perform a custom fetch, attach logic to this event to fetch a single row. To indicate that there are more records to fetch, use the system function, **Continue Custom Data Fetch** (set it to True).

When runtime loads the grid control, it fires **Add Last Entry Row to Grid** when (and if) it adds an entry row as the last line in the grid.

The pattern for adding and updating records is to fire an event immediately before and after commit. Then, runtime fires another event after all the records have been processed. This is a list of applicable events:

- 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

The pattern for deleting a row is similar except that runtime notifies you before and after the user action as well as the commits:

- Delete Grid Rec Verify - Before
- Delete Grid Rec Verify - After
- Delete Grid Rec from DB - Before
- Delete Grid Rec from DB - After
- All Grid Recs Deleted from DB

Runtime notifies you of these user actions:

- **Column Selection Changed**

This event applies to check box grid columns only. **Column Selection Changed** fires if the user selects or clears an editable check box grid cell. It does not fire if the grid cell is modified by event rules.

- **Grid Column Clicked**

This event fires only if the grid column was configured as being clickable during design time.

- **Visual Assist Button Clicked and Post Visual Assist Clicked**

If you want to override the default UDC form that appears automatically in response to clicking the Visual Assist, these are the events to which to add the logic.

- **Aggregation Clicked**

This event applies to Support Aggregation column property and allows you to override the UDC form.

- **Double Click on Row Header**

Finally, runtime signals as the user works through the grid, starting when the grid gets focus and ending when the grid loses focus.

- **Set Focus on Grid**

This event fires whether the user or runtime changes the focus.

- **Grid Cell Display Changed**

This event applies only to icon grid columns (those for which the Display Style property has been set to Icon.) The event fires and the cell's display icon setting changes whenever the value of the data in the cell changes. The value can change by fetching data, adding or updating records in a grid, setting the output parameter of a system or business function call, or by explicit ER assignment. It can therefore fire at many different times during the grid population, navigation and editing life cycle, including within other grid events.

This event should reference a client-side NER, which internally calls the **Set Grid Cell Icon** system function, which sets the new icon or tooltip to display for the column. You should make only a simple, single reference to the system function. If you do not call this system function, the cell displays a blank image.

- **Column Selection Changed**

This event fires only if the column is a check box column (that is, the column Display Style-Default or Checkbox property has been set to **Checkbox**).

- **Row is Entered**

- **Row is Selected (Web Only)**

This event fires on power forms and subforms only.

- **Row is Exited**

This event fires whenever a row is exited.

- **Row Exit & Changed - Inline**

This event fires after **Row is Exited** if the grid is an update grid and the row has been changed since the last time the row was exited.

The Trigger Parallel Event system function is available for the **Row Is Exited and Changed** and **Row Is Exited and Changed In-Line** events. This system function will enable a parallel event to run on a separate thread and will not interfere with existing Event Rules.

- **Row is Exited & Changed - Asynch**

This event fires after **Row Exit & Changed - Inline** if the grid is losing focus, or if another row is entered. **Row is Exited & Changed - Asynch** is equivalent to

validating the contents of the row. It is often used for the Edit Line master business function.

- **Kill Focus on Grid**

This event fires whether the user or runtime changes the focus.

21.4 Grid Control Runtime Processing

This section discusses runtime processing for the grid control.

21.4.1 How Runtime Processes the Grid Control

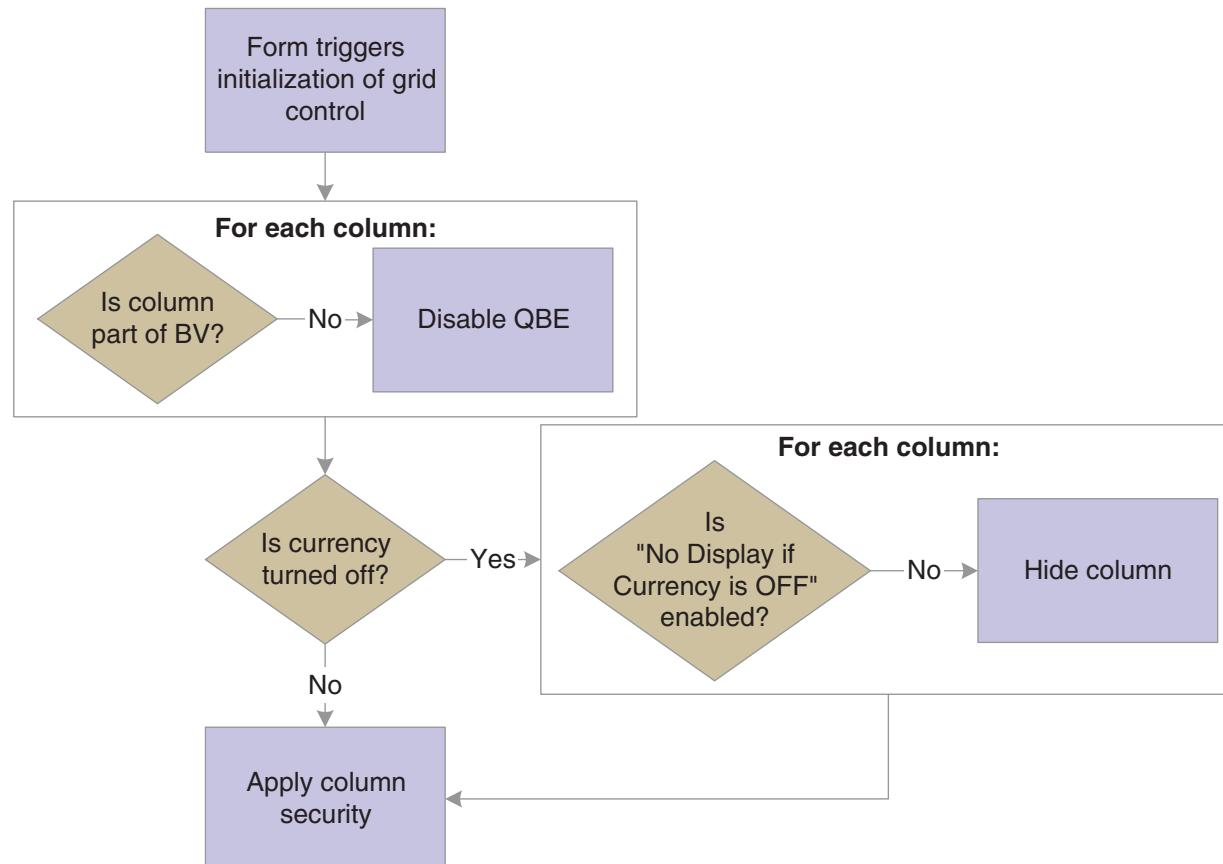
This subsection discusses how runtime processes the grid control.

Typically, runtime processes the grid control at these points in a standard workflow:

- On control initialization (triggered by form initialization).
- On grid population request (triggered conditionally by control initialization, Next, and Previous, and always by Find).
- On row or control exit.

This flowchart illustrates the initialization process for a grid control:

Figure 21–1 Grid control initialization

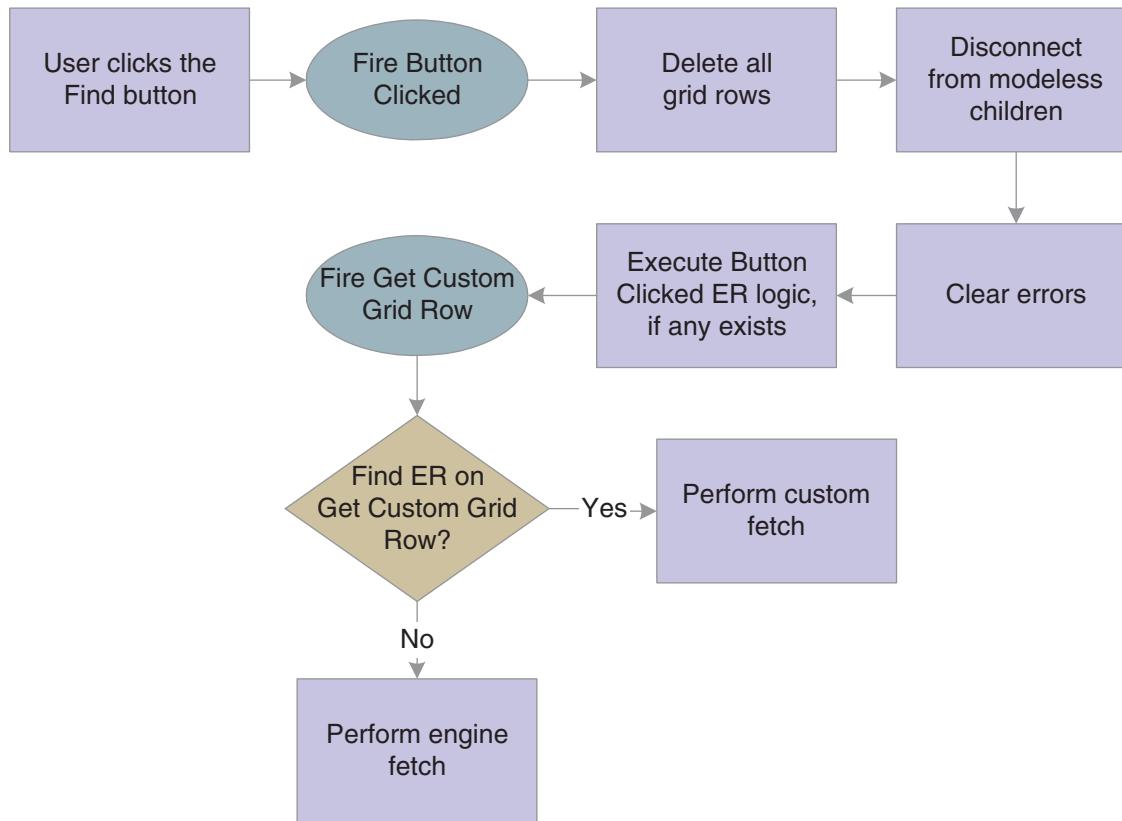


Grid population occurs immediately after the initialization process if the option, Automatically Find On Entry is selected. The grid population process is also triggered

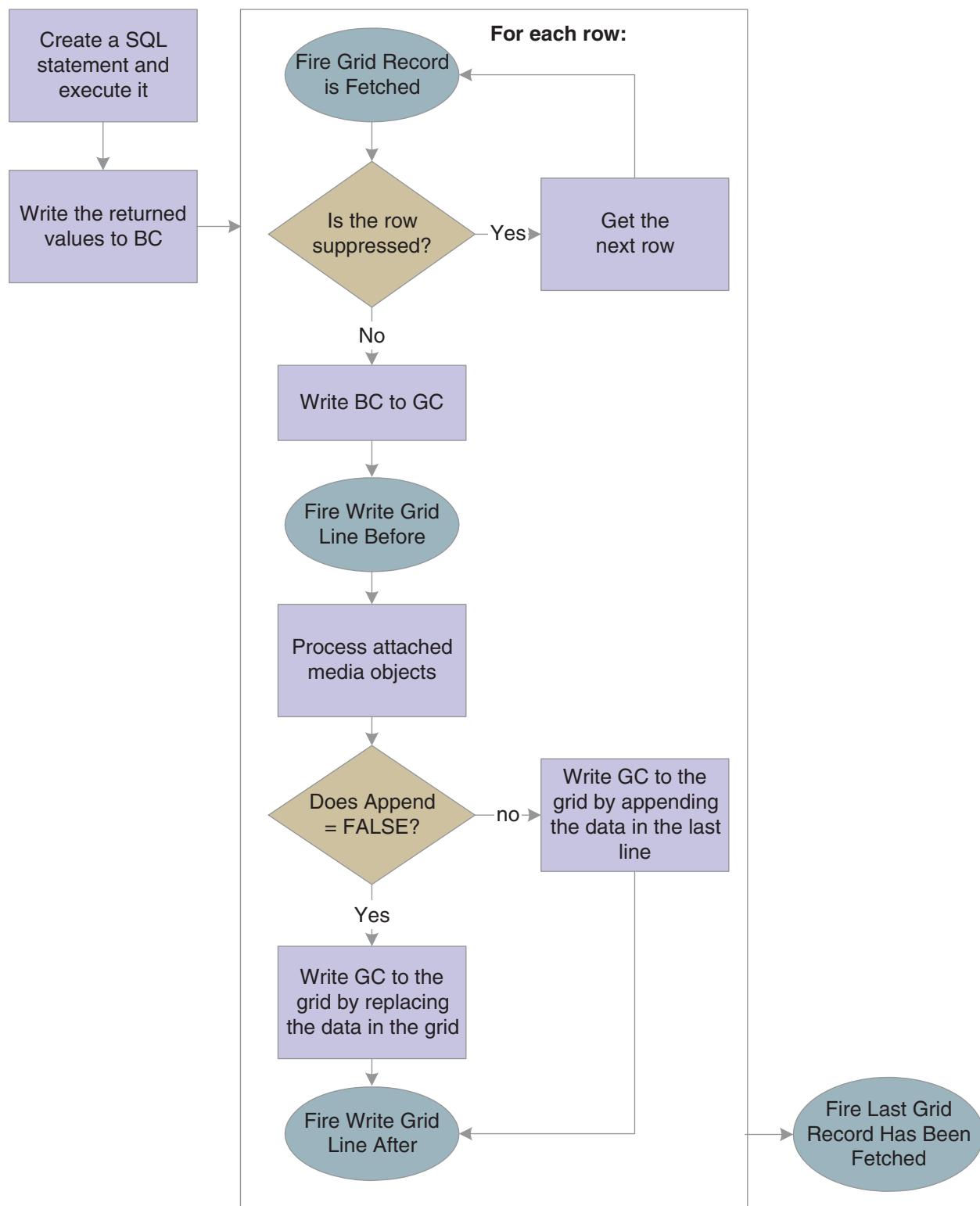
when the user clicks the Find button or if you apply logic on an event to do so. If the fetch requires access to the form's BV only, then you can probably enable the engine to perform the fetch on its own. However, if you want to access other tables, you must set up and execute a custom fetch programmatically.

This flowchart illustrates the steps that occur when find is initiated:

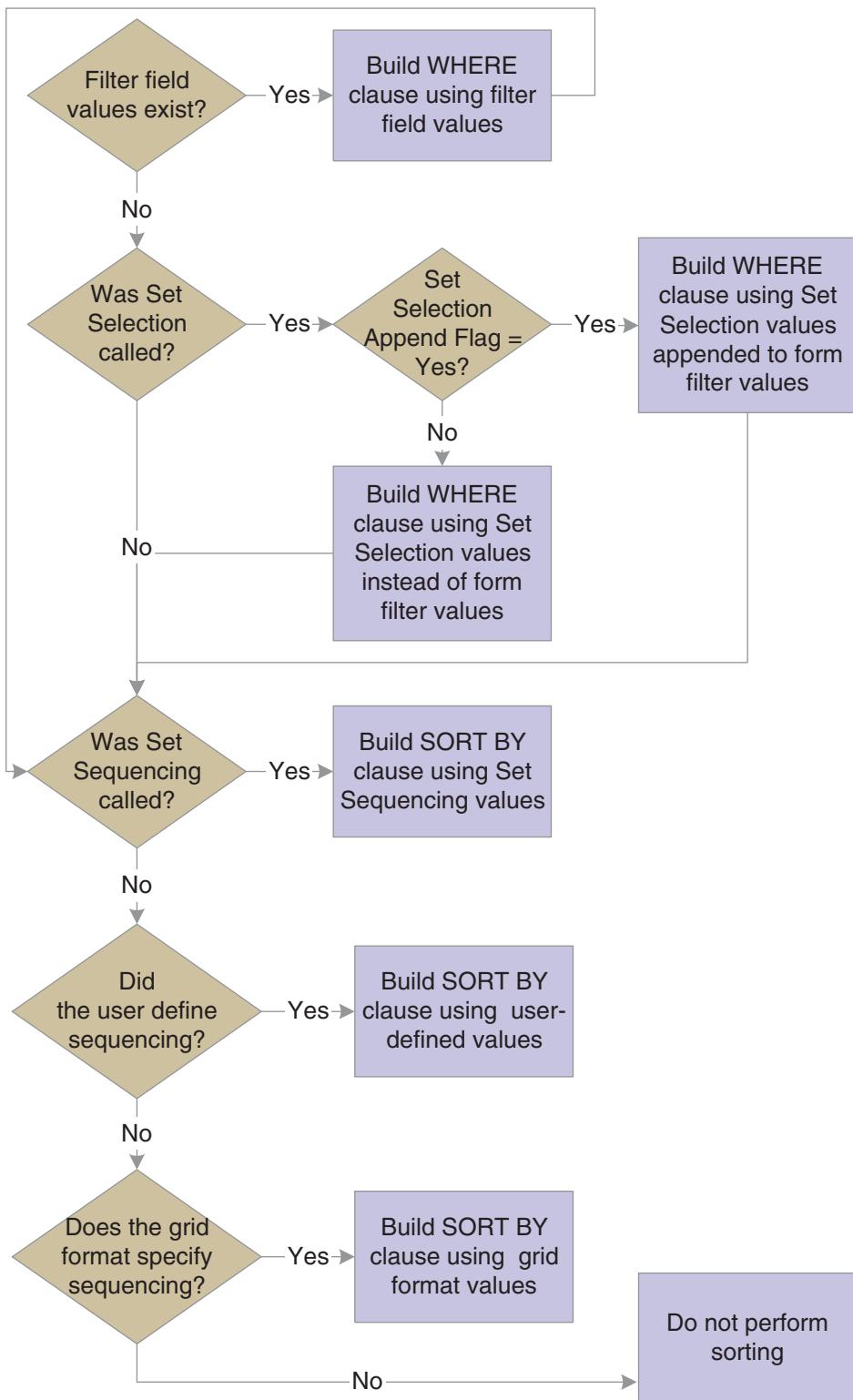
Figure 21–2 Grid control find button processing



This flowchart illustrates how the system performs an engine fetch:

Figure 21–3 Grid control fetch processing

This flowchart illustrates how the system creates the SQL statement:

Figure 21–4 Grid control SQL processing

Custom fetches are executed based exclusively on the ER associated with the **Get Custom Grid Row** event. The ER provides instructions for how to load a row, and runtime repeats the ER until you tell it to stop or it reaches the end of a page if page-at-a-time processing is enabled. To tell the engine to continue looping or not,

include a call to **Continue Custom Fetch** in the ER. Set **Continue Custom Fetch** to True to run another loop; set it to False to exit the custom fetch loop. (Not calling **Continue Custom Fetch** is equivalent to setting it to false.)

Grid controls are used to display information. In some instances, information display is all the grid does, in which case, no more grid control processing occurs.

Grids can also be used to enable users to delete, edit, and add data as well. To enable any of these functions, the grid (and therefore the form) must be in Add or Update mode (Update mode is the default mode; add mode occurs either when the user clicks Add or a fetch fails to retrieve data). Furthermore for add and update capabilities, the grid must not be disabled; that is, the Disable Input option was not set at design time. Finally, to add data, the user must have an entry row (that is, a blank row) as well. Runtime automatically adds an entry row to update grids in the add and update modes as long as the No Adds on Update Grid option was not set at design time.

When the user tabs out of an entry row, runtime performs these functions in this order:

1. Clear the GC hash table.
2. Fire **Last Entry Row to Grid**.
3. Get the value from the GC and copy the contents into a new entry row.

This three-part flowchart illustrates how runtime processes the act of writing data from the grid and committing the change to the database:

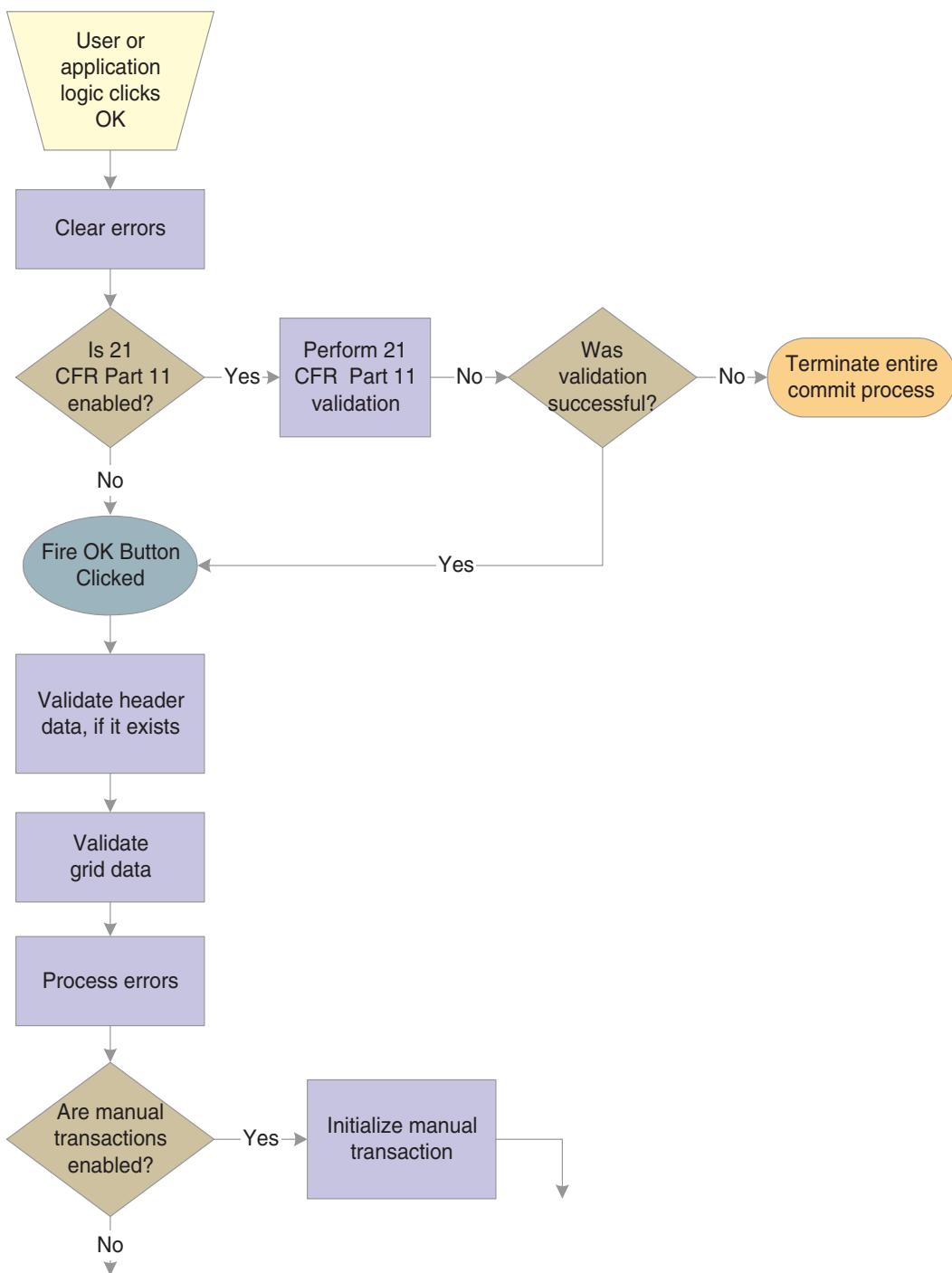
Figure 21–5 Grid control database commit, part 1 of 3

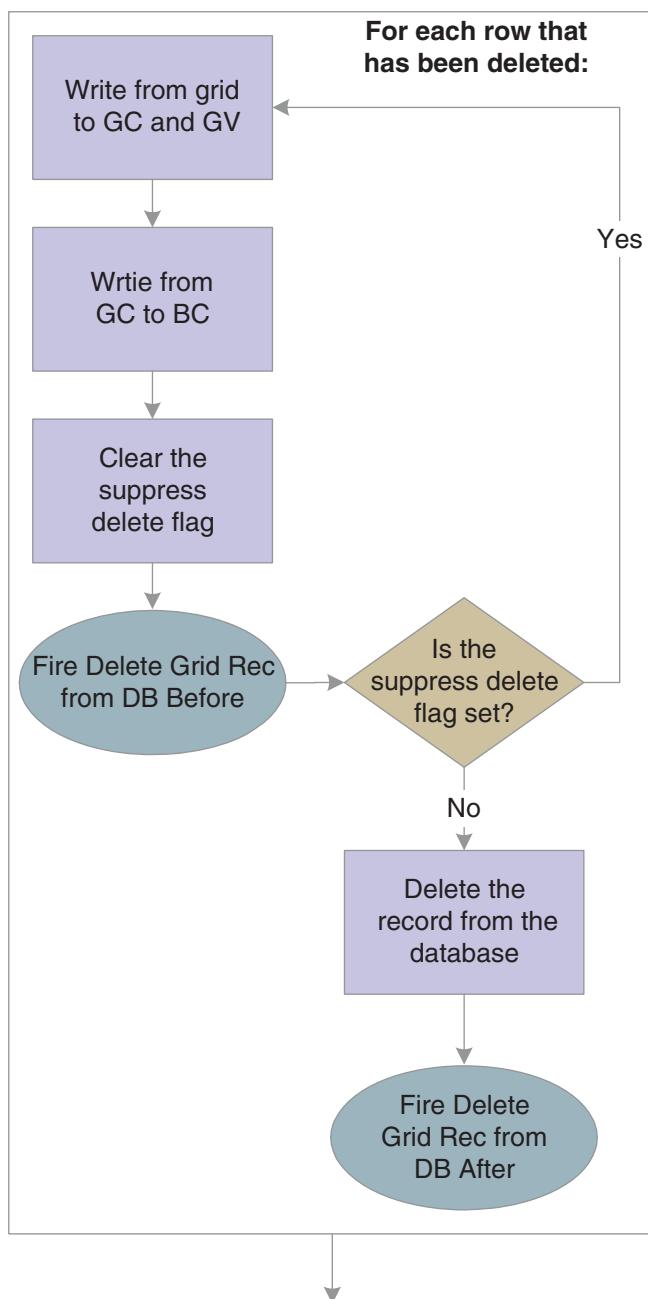
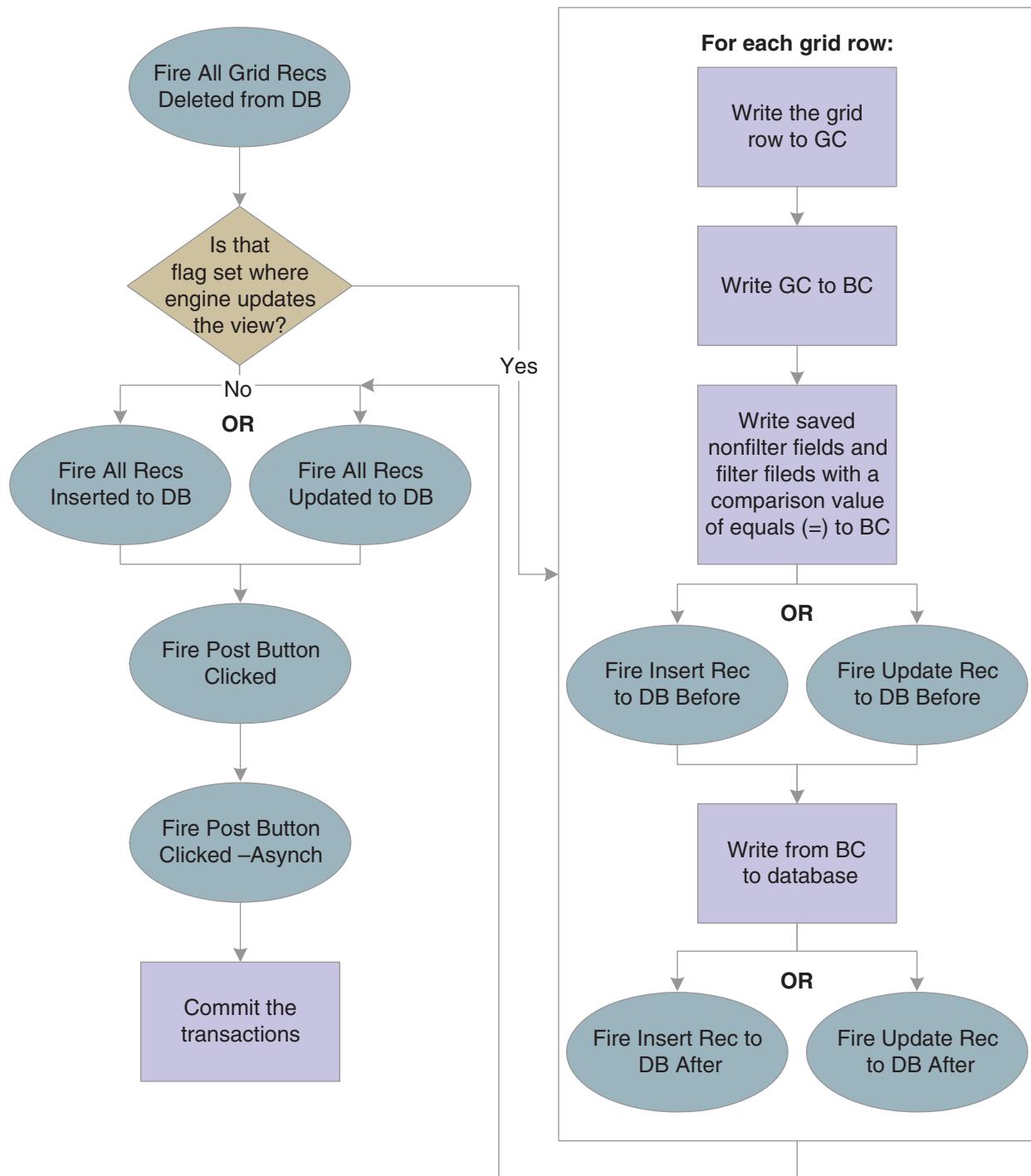
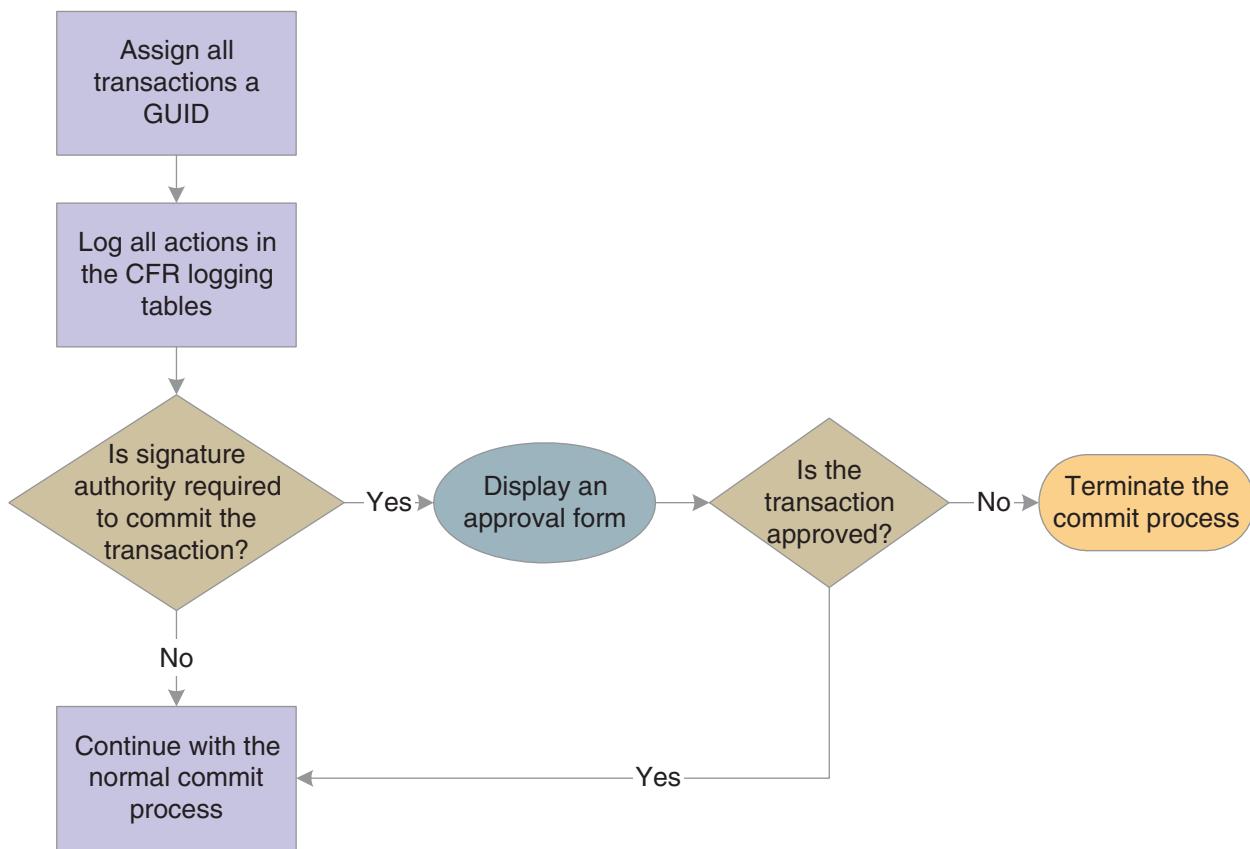
Figure 21–6 Grid control database commit, part 2 of 3

Figure 21–7 Grid control database commit, part 3 of 3



This flowchart illustrates 21 CFR 11 validation:

Figure 21–8 Grid control 21 CFR validation

This two-part flowchart illustrates grid data validation:

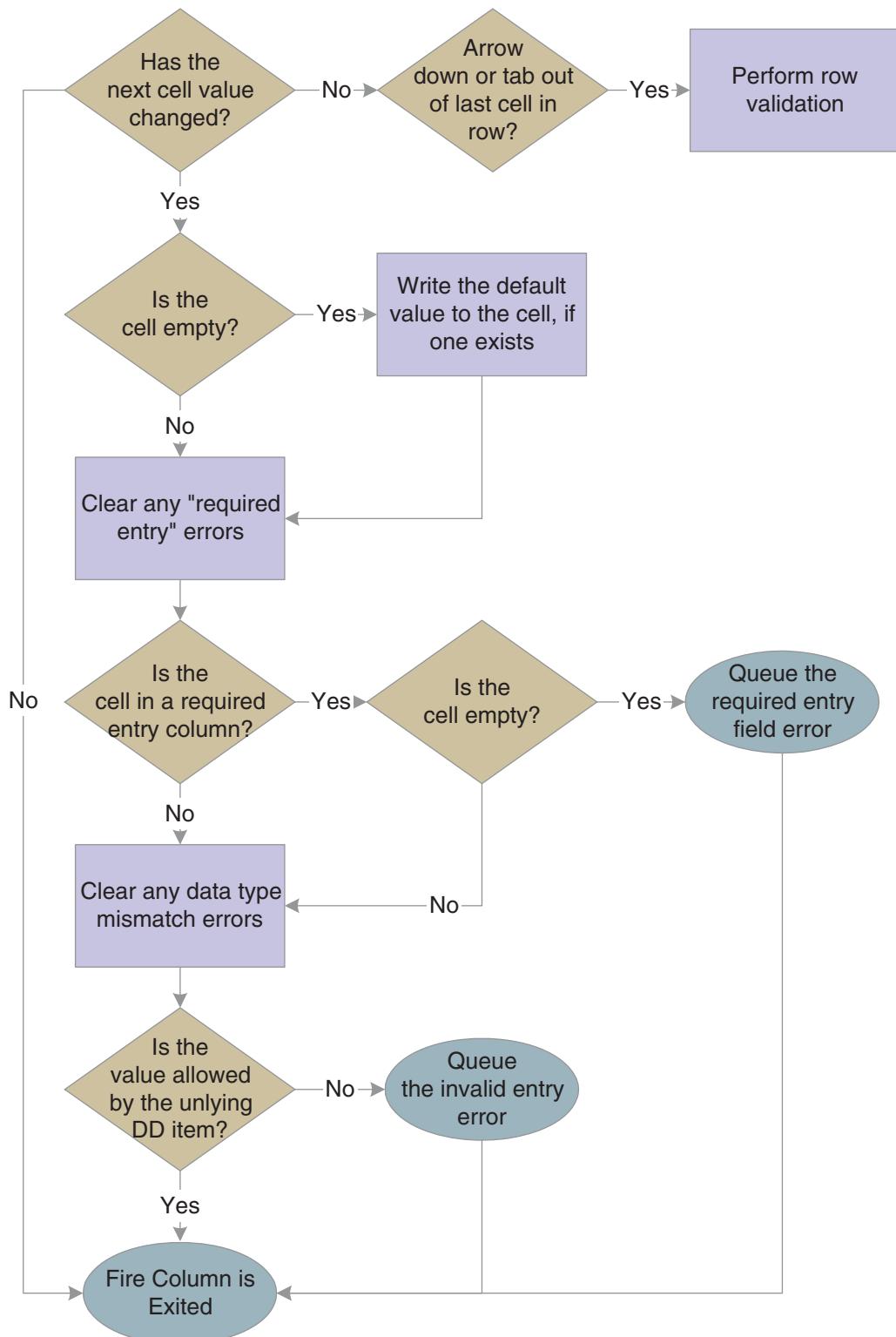
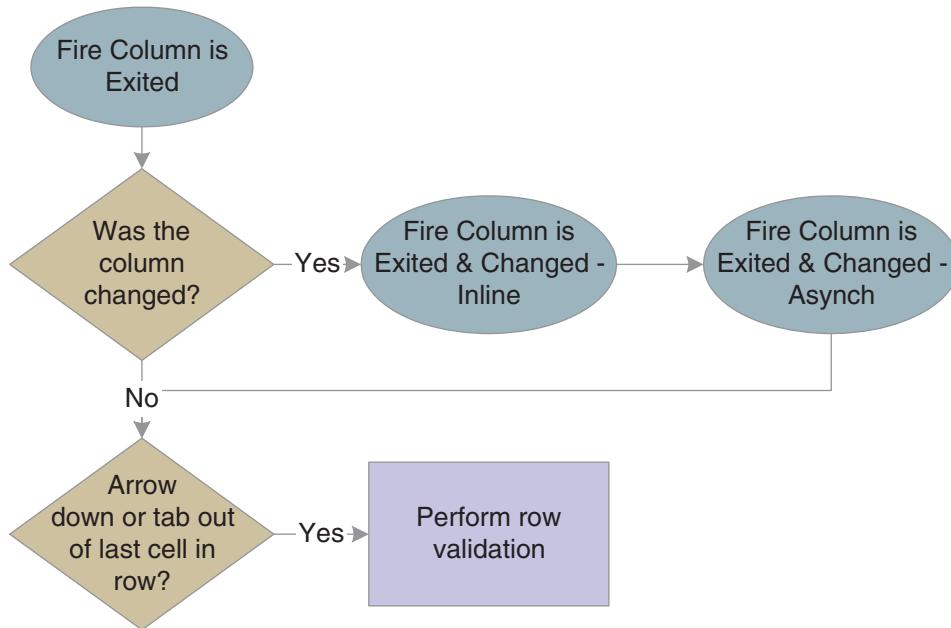
Figure 21–9 Grid control data validation, part 1 of 2

Figure 21–10 Grid control data validation, part 2 of 2

21.4.2 Impact of Interactivity Levels

This subsection discusses the effect of different runtime processing environments and modes: HTML (low or high interactivity) or Windows.

This table contrasts the differences:

Grid Type	Interactivity	Effect
Non-editable	Low	Refresh the entire form (including grid control) on Find, Next, and Previous. Display only one page of data in the grid.
Non-editable	High	Refresh only the grid (not the entire form) on Find. Append additional rows and expand the page size on Next. The interface provides a vertical scroll bar.
Non-editable	Windows	Refresh only the grid (not the entire form) on Find.

Grid Type	Interactivity	Effect
Editable	Low	<p>Display only one page of data in the grid. Post and refresh behavior depends on whether the Multi-Line Edit option is enabled:</p> <ul style="list-style-type: none"> ▪ Multi-Line Edit option disabled: Post inline on row exit and refresh the whole form. ▪ Multi-Line Edit option enabled: Post asynchronously and refresh the grid rows every 3-5 row exits.
Editable	High	<p>Expand the page and add a vertical scroll bar to display all rows. Post asynchronously and refresh the grid control at every user action (row exit, column exit, and so forth). Add a new entry row upon user keying into and exiting any cell in the entry row.</p>
Editable	Windows	<p>Expand the page and add a vertical scroll bar to display all rows. Process every user action immediately. Add a new entry row upon user keying into any cell in the entry row.</p>

For high interactive editable grid, post asynchronously for each cell change.

21.5 Grid Control System Functions

This section discusses the system functions you can use to affect how the grid control works during runtime.

Grid system functions enable you to manipulate the appearance of the grid and what goes into or comes out of it during runtime. In the latter case, the system applies the command either to the model version of the grid or to the display version. The model version of the grid contains all rows and columns, whereas the display version might have hidden rows or columns. This is an important distinction to make when you are performing actions based on row or column number. For example, assume a grid contains three rows, but the second one is hidden. If you insert a row beneath row number two in the model grid, the insert row becomes row three, and row three becomes row four. However, if you insert a row beneath row number two in the display grid, the row is inserted at the bottom of the grid, because the actual row number two is hidden.

Another distinction to make is whether the system function expects the first grid row number to be zero or one. All system function descriptions where these factors might be an issue include whether they affect the model or display version of the grid and whether they are zero- or one-based.

Similar to the Table View grid format, the List View grid format supports all the grid control system functions, except the Set Grid Font system function. (Release 9.2.0.5)

Some grid-related system functions enable you to alter the appearance of the grid. Use **Set Grid Color** to change the background color of a cell, row, column, or the entire control. Use **Set Grid Font** to change the type, style, size, effects, and color of the font in a cell, row, column, or the entire control. Alternatively you can use **Set Grid Row Format** to use HTML formatting commands to control the appearance of the grid, not only for color and font, but for other factors as well. Use **Set Grid Cell Icon** within a client-side, icon-setting NER to set an icon for a grid cell. Use **Set Grid Cell Icon Visibility** within normal form-based ER to control an icon's visibility dynamically.

To enable or prevent a user from customizing the grid, use **Display Customize Grid Option**.

Use **Hide Grid Column** and **Show Grid Column** to hide and display columns in the grid. Although a user cannot affect a hidden column, you can still work with and change values in it programmatically.

If the column is based on a DD item, use **Set Data Dictionary Overrides** to change the DD attributes of the column, or **Set Data Dictionary Item** to substitute another DD item altogether. To change the heading for the column, use **Set Grid Column Heading**.

Use **Hide Grid Row** and **Show Grid Row** to hide and display rows in the grid. As with columns, you can still affect hidden grid rows programmatically.

You can also control the bitmap icon that appears next to a given row with **Set Grid Row Bitmap**.

You can enable or prevent users from changing an existing cell, row, column, or the entire control. Use **Enable Grid** to enable user edits (assuming that the current mode permits such actions) and **Disable Grid** to prevent them. Although users cannot alter a disabled grid object, you can change it programmatically.

You can enable or prevent users from importing from Excel or from importing or exporting to Excel or Word with **Display Import from Excel Option**, **Display Export to Excel Option**, and **Display Export to Word Option**.

You can determine if a grid cell has been updated since the last check with **Was Grid Cell Value Entered**.

Use **Get Max Grid Rows** to count the number of rows currently loaded into the grid control. Use **Get Selected Grid Row Count** to count the number of rows in the current selection. The selected area need not consist of contiguous rows.

Use **Set QBE Column Compare Style** to set the comparison value (=, >, <=, and so forth) for a QBE column. **Clear QBE Column** clears an entry in a QBE column.

Runtime always fetches all records for all columns defined by the BV. However, you can impact how runtime builds the WHERE and ORDER BY clauses of the SQL SELECT statement it executes to perform a fetch. (Use the various Hide system functions to filter what the user sees even more, if necessary.) Use **Set Selection** to input the specific values you want to use in the WHERE clause. For example, assume that you want a list of accounts that are greater than 1000 USD. The SQL WHERE statement might look like this (literal values are used to simplify the examples):

```
WHERE F0902.GBAN01 > 1000
```

The **Set Selection** statement that builds this code would look like this:

```
Set Selection(FC Grid, F0902, GBAN01, <Greater Than>, '1000', <None>)
```

That is: (grid control, table, alias, comparison type, comparison value, and/or).

The values you input with **Set Selection** persist until form close. Consequently, subsequent calls to **Set Selection** enable you to build a complex WHERE clause. For

example, say you wanted a list of document types for Foundation. In the F0005 table, system code and object type are in two different columns, so you must call **Set Selection** twice. First:

```
Set Selection(FC Grid, F0005, DRSY, <Equal To>, '00', <None>)
```

Then:

```
Set Selection(FC Grid, F0005, DRRT, <Equal To>, 'DT', <AND>)
```

By setting the and/or parameter to **<AND>**, the system knows to append the WHERE clause with the values from a subsequent call to **Set Selection** with an AND. Therefore, runtime will build this WHERE clause:

```
WHERE F0005.DRSY = '00' AND F0005.DRRT = 'DT'
```

Use **Set Selection Append Flag** to indicate whether the criteria from the **Set Selection** system function should replace the criteria from existing filter fields and QBE or be appended to it.

You can alter the sort order of the returned values by using **Set Sequencing**. **Set Sequencing** builds the ORDER BY clause of the SQL statement. The **Set Sequencing** parameters are: Set Sequencing (grid control, table, alias, sort order), so if we wanted to sort the preceding example by the document type code (DRKY) in ascending order, you would code:

```
Set Selection(FC Grid, F0005, DRKY, <ASCENDING>)
```

If you need to build a WHERE clause and do not want to use values that may already exist because of previous uses of **Set Selection**, use **Clear Selection** to delete all existing selection (but not sequencing) values from memory. To remove sequencing values, use **Clear Sequencing**.

When you fetch data, you can populate GB and then use system functions to write GB values to the grid control. You can also write from the grid control to the GB and manipulate the GB and grid data in other ways.

To write from the GB to the grid, use either **Insert Grid Buffer Row** or **Update Grid Buffer Row**, depending on whether you want to add a new row to the grid (insert) or overwrite an existing row (update). In either case, when you write a new row, you have a variety of configuration options. You can make the new row selectable, editable, updatable, and deletable, assuming that the current conditions permit such actions. You can also clear the GB automatically after insert so that you do not need to make repeated calls to **Clear Grid Buffer**, which is the system function that removes all values from the GB.

Insert Grid Buffer Row and **Update Grid Buffer Row** both require a row number that is based on the model grid as input so that runtime knows where to place the new row. One way to determine a grid row number is with the system function, **Get Selected Grid Row Number**.

To activate a specific grid row during an event, use **Get Grid Row**. After identifying a row with **Get Grid Row**, all subsequent ER that normally references GCs will reference the identified row instead. The alternate reference lasts for the duration of the event.

To prevent a row from being written to the grid during the runtime fetch, use **SUPPRESS GRID LINE**.

To copy or populate a grid row to the GB, use **COPY GRID ROW TO GRID BUFFER**. To remove a grid row from the grid control, use **DELETE GRID ROW**.

You can set or clear an error on a cell, row, column, or the entire control. Use **Set Grid Cell Error** to set errors and **Clear Grid Cell Error** to clear them.

Change Row Selection

Use this system function to select or deselect grid rows programmatically.

Parameters

Grid

Input, required. The grid form control (FC) to affect.

Row

Input, required. The row to affect. Set the parameter to **<All Rows>**, **<Currently Selected Row>**, or an applicable object from the object list.

Select State

Input, required. The state (selected or deselected) in which to put the indicated row or rows. Set the parameter to **<Selected (1)>**, **<Unselected (0)>**, or an applicable object from the object list.

Clear Grid Buffer

Use this system function to clear the grid buffer (GB) manually.

Parameter

Grid

Input, required. The grid FC to affect.

Clear Grid Cell Error

Use this system function to clear an error on a cell, a row, a column, or the entire control. Also, this system function does not take hidden rows into account; in other words, it affects the display grid instead of the model grid.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The row in which to clear the error. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column in which to clear the error. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

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

Clear QBE Column

Use this system function to clear text from a single QBE column or from all columns.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The QBE column to clear. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Additional Notes

You might use this function when the QBE column had a value forced into it, such as the current date on a purchase order find/browse form. Then, after the user performs a Find, you could clear the QBE column in case you want to use another date.

Clear Selection

Use this system function to clear all system-function-defined selection information that was used to build previous SQL statements during the current session.

Parameter

Grid

Input, required. The grid FC to affect.

See Also: [Set Selection](#)

Clear Sequencing

Use this system function to clear all system-function-defined sequencing information that was used to build previous SQL statements during the current session.

Parameter

Grid

Input, required. The grid FC to affect.

Copy Grid Row to Grid Buffer

Use this system function to write all of the GC fields to GB. Depending on the conditions when this function is initiated, the GC can be either in memory or in the grid. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The row to copy to GC. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>, or applicable object from the object list.

Delete Grid Row

Use this function to delete a grid row from the model grid. That is, you can use this system function to affect hidden rows. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The row to delete. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>, or applicable object from the object list.

Disable Grid

Use this system function to protect a cell, a row, a column, or the entire control from being edited.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to disable. The first row is 0. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column to disable. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Additional Notes

This function is useful when you need to complete more information in the header before you can add individual rows or when you want to use information in the header as default information in the grid. Also, this system function does not take hidden rows into account when deleting a row based on an absolute row number value.

See Also: [Enable Grid](#)

Display Customized Grid Option

Use this function to hide or display the option that enables users to customize the grid. This includes the customize grid link on the grid header and the **Grid Formats** menu item under Tools menu.

Parameters

Grid

Input, required. The grid FC to affect.

Enable

Input, required. A flag indicating whether to hide (Enable=<FALSE>) or display (Enable=<TRUE>) the option. Set the parameter to <TRUE> or <FALSE>.

Display Export to Excel Option

Use this function to hide or display the option that enables users to export the contents of the grid control to Excel. This includes the "Export to Excel" button on the grid header and the "Export To Excel" menu item under Tools menu.

Parameters

Grid

Input, required. The grid FC to affect.

Enable

Input, required. A flag indicating whether to hide (Enable=<FALSE>) or display (Enable=<TRUE>) the option. Set the parameter to <TRUE> or <FALSE>.

See Also: [Display Export to Word Option](#)

[Display Import from Excel Option](#)

Display Export to Word Option

Use this function to hide or display the option that enables users to export the contents of the grid control to Word. This includes the **Export to Word** button on the grid header and the **Export To Word** menu item under Tools menu.

Parameters

Grid

Input, required. The grid FC to affect.

Enable

Input, required. A flag indicating whether to hide (Enable=<FALSE>) or display (Enable=<TRUE>) the option. Set the parameter to <TRUE> or <FALSE>.

Tip:

[Display Export to Excel Option](#)

[Display Import from Excel Option](#)

Display Import from Excel Option

Use this function to hide or display the option that enables users to import the contents of an Excel file into a grid. This includes the "Import From Excel" button on the grid header and the "Import From Excel" menu item under Tools menu.

Parameters

Grid

Input, required. The grid FC to affect.

Enable

Input, required. A flag indicating whether to hide (Enable=<FALSE>) or display (Enable=<TRUE>) the option. Set the parameter to <TRUE> or <FALSE>.

See Also: [Display Export to Excel Option](#)

[Display Export to Word Option](#)

Enable Grid

Use this system function to enable a user to edit a cell, a row, a column, or the entire grid. Also, this system function does not take hidden rows into account when enabling a row based on an absolute row number value.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to enable. The first row is 0. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column to enable. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

See Also: [Disable Grid](#)

Get Grid Row

Use this function to target a grid row for use in an upcoming function in place of the grid row that would be used ordinarily. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to acquire. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list.

Additional Notes

If the row specified is greater than the total number of rows, the last row is used. If the row specified is invalid, the active row becomes zero.

Get Max Grid Rows

Use this system function to count the number of rows in the model grid; that is, it counts both hidden and visible rows.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the number of rows in the model grid to the object identified by Row.

See Also: [Disable Grid](#)

Get Next Selected Row

Use this system function to determine the next selected grid row after a given row.

Parameters

Grid

Input, required. The grid FC to affect.

Start Row

Input, required. The row from which to start searching for a selected row. In other words, runtime will return the first row it finds, after this row, that has been selected. To include the first row of the grid in the search, select the special parameter, <Before First Row>. Set the parameter to <Before First Row> or an applicable object from the object list.

Row

Output, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the index of the first selected row the system encounters beneath the indicated row. Runtime writes the index to the object specified by Row.

Get Selected Grid Row Count

Use this function to count the number of grid rows in the current selection. The rows to be counted need not be contiguous.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Output, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the number of rows in the current selection to the object identified by Row.

See Also: [Get Max Grid Rows](#)

Get Selected Grid Row Number

Use this function to get the row number for a selected row. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Output, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Additional Notes

If multiple rows are selected, the function returns the index value for the first row. Typically, you use this function only when you need to save the row as a variable for future processing.

Returns

This system function returns the index position of the selected row to the object identified by Row.

Hide Grid Column

Use this function to prevent an entire column from appearing on the form.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column to hide. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

See Also: [Show Grid Column](#)

Hide Grid Row

Use this system function to prevent an entire row from appearing on the form. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to hide. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, an applicable object from the object list.

Additional Notes

This is an example of how to use this system function. If you create a form with a grid and you want to summarize rows, add together several rows to determine a total and show the total row. If you use this system function, you might include a check box for the detail information and then display the row instead of repopulating the grid.

If no row is selected when it is invoked, then the system function hides the last row.

See Also: [Show Grid Row](#)

Insert Grid Buffer Row

Use this system function to insert a row from the GB into the grid control. After the grid row is inserted, the row will receive focus when the grid is updated. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to insert the row. Set the parameter to an alphanumeric constant (<Literal>), <After Current Row>, or <After Last Row>.

Selectable?

Input, required. An indicator of whether the Select button processes for the inserted row. Set the parameter to <Yes> or <No>.

Protected?

Input, required. An indicator of whether the user can edit the inserted row. Set the parameter to <Yes> or <No>.

Updateable?

Input, required. An indicator of whether runtime will attempt to update or insert the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable?

Input, required. An indicator of whether the user can delete the inserted row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the insert. Set the parameter to <Yes> or <No>.

See Also: [Update Grid Buffer Row](#)

Insert Grid Row

This system function is deprecated. Use **Insert Grid Buffer Row** instead.

See [Insert Grid Buffer Row](#).

Set Data Dictionary Item

Use this system function to override form controls and grid columns that are DD items. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column to change. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

DD Alias

Input, required. The alias of the DD item that you want to use. Set the parameter to a specific DD alias (<Pick DD Item>) or an applicable object from the object list.

System Code

This parameter is reserved for future functionality.

Additional Notes

This system function makes a complete change; you substitute a different DD item for the existing one. You can also create a new data item that is not the same type as the old item. The new DD item must be the same data type as the previous DD item or the system function call does not make any changes.

Set Data Dictionary Item Overrides

Use this system function to change a specific DD item property. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column to override. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Overrides

Input, required. The override to apply. Set the parameter to the specific type of override to apply.

Set Grid Cell Error

Use this system function to set an error on a cell, a row, a column, or the all columns in the grid. Also, this system function does not take hidden rows into account; in other words, it affects the display grid instead of the model grid.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row on which to set the error. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, or applicable object from the object list.

Column

Input, required. The column on which to set the error. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Error Code

Input, required. The alias of the error to be set on the row and column. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list.

See Also: [Clear Grid Cell Error](#)

Set Grid Cell Icon

Use this system function to set the icon and the icon-specific tooltip text to display in the calling grid cell. This system function only processes when called from the **Grid Cell Display Changed** event of a grid column. The grid cell location information is calculated by runtime from the context of the event in which the function was called, and is therefore not explicitly defined in this function. This function has no effect when invoked from any other event, and is only available when defining a NER.

To ensure that translations are in effect for icon-specific tooltip text, you should use either the Define Message special value to implement text substitution on DD glossary text, or UDC text with locally-defined variables.

Parameters

Image File

Input, required. The name of the reference image to be rendered in the calling grid cell. Set the parameter to None, or use the Image Picker to see and select the available icon reference images in the current H4A (or WLSH4A) installation.

Icon Tooltip

Input, required. The tooltip text associated with selection of this icon. Set the parameter to an applicable object (such as a NER variable), to None, or Define Message using the template Text Substitution mechanism.

Set Grid Cell Icon Visibility

Use this system function to show or hide an icon on a cell, a row, a column, or all columns in the grid. This system function does not take hidden rows into account. It affects the display grid instead of the model grid. This function has no effect on grid cells that are not rendered as icons.

Parameters

Grid

Input, required. The grid to affect

Row

Input, required. The row in which to show or hide icons.

Column

Input, required. The column in which to show or hide icons.

Visibility

Input, required. The visibility state to apply to icons.

Set Grid Color

Use this system function to set a color on a cell, a row, a column, or all columns in the grid. You can select a specific color to set, or you can reset the color which returns the color of the object to its default value. The system function affects the model version of the grid and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row on which to set the color. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, or applicable object from the object list.

Column

Input, required. The column on which to set the color. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Color

Input, required. The color to be set on the row and column. A color from the color palette (<Pick Color>) or the default color value (<Reset Color>).

Set Grid Column Heading

Use this system function to change the text in a grid column heading. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column for which to change the header. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Text

Input, required. The text to use for the column header. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list

See Also: [Set Data Dictionary Item Overrides](#)

Set Grid Font

Use this system function to set a font for the text in a cell, a row, a column, or all columns in the grid. In this context, font includes typeface (such as Arial or Times New Roman), style (such as italic or bold), size, effects (strikeout or underline), and color. You can select a specific font to set, or you can reset the font which returns the font of the object to its default value. This system function affects the model version of the grid, and it is one-based.

Note: The List View grid format does not support the Set Grid Font system function. (Release 9.2.0.5)

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row for which to set the font. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, or applicable object from the object list.

Column

Input, required. The column on which to set the font. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Font

Input, required. The font to be set on the row and column. Set the parameter to a font and related settings from the Font dialog (<Pick Font>) or the default font settings (<Reset Font>).

Set Grid Row Bitmap

Use this system function to manipulate the bitmap icon that appears next to the row. This function sets a specific system bitmap, such as a check mark or a trash can, to use as an icon on a specified row header. You cannot modify the icon list. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row for which to set the bitmap. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, or applicable object from the object list.

Bitmap

Input, required. The bitmap to be set on the row. Set the parameter to the specific bitmap that you want to apply (<Checkbox>, <X Mark>, and so forth).

Set Grid Row Format

This system function applies a system-defined HTML row format or returns to the default value.

Parameters

Grid

Input, required. The grid FC to affect.

Format

Input, required. The format to apply. You can apply the default format, or you can use the alternate HTML formatting string you entered into the Alternate Grid Row Format String parameter for the grid at design-time. Set the parameter to <Default> or <Alternate>.

Set Lower Limit

Use this system function to create the WHERE clause of a SQL search statement for use in a succeeding fetch when you are searching against a table that employs a ragged hierarchy.

Parameters

Grid

Input, required. The grid FC to affect.

Table

Input, required. The source from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Alias

Input, required. The column of the table from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Comparison Type

Input, required. The comparison operator to use. Set the parameter to the comparison operator to use (<Equal To>, <Not Equal To>, and so forth).

Comparison Value

Input, required. The comparison value to use. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list.

And/Or

Input, required. Indicates whether to append another WHERE clause to the SQL statement and how the new statement relates to the one before it (AND or OR). Set the parameter to <And>, <Or> or <None>.

Additional Notes

In some cases, referencing an item just by table and column (as you do with **Set Selection**) is insufficient. For example, in JD Edwards EnterpriseOne, some tables, especially in Financials, are structured using ragged hierarchies. A ragged hierarchy is one in which the parent attribute of one or more child attributes is not at the level immediately above the child. In short, some attributes in the hierarchy effectively have an empty parent-level attribute and descend from a grandparent attribute instead.

This table illustrates how information might appear in a ragged hierarchy:

Segment	Family	Class	Commodity
Mineral & Textile			
Mineral & Textile	Mineral, Ores, & Metals		
Mineral & Textile	Mineral, Ores, & Metals	Base Metals	
Mineral & Textile	Mineral, Ores, & Metals	Base Metals	Titanium

Segment	Family	Class	Commodity
Mineral & Textile	Mineral, Ores, & Metals	Precious Metals	
Mineral & Textile	Mineral, Ores, & Metals	Precious Metals	Gold
Time, Jewelry, & Gem			
Time, Jewelry, & Gem	Gemstone		
Time, Jewelry, & Gem	Gemstone	Pearls	
Time, Jewelry, & Gem	Timepiece		
Time, Jewelry, & Gem	Timepiece	Watches	

If you searched at the commodity level, the system would return only titanium and gold because at that level, watches and pearls are technically null values. To compensate, use the system function, **Set Lower Limit**, instead of **Set Selection** to build the WHERE clause of the SQL statement in this situation. With **Set Lower Limit**, you can have the grid display all records starting from the Titanium line down. With conventional filter fields it is impossible to display all the subsequent records because none of the other "class" fields are greater than or equal to Titanium. **Set Lower Limit** not only includes the most restrictive field (in this case Class - Titanium), but also includes each step in the hierarchy to determine if a record is actually "greater than" the lower limit record.

In general, this system function should not be used in cases where the user can affect the select. For example, QBE and filter fields are ways the user can affect the select. If the user can affect the select, then the user's select information should always be taken into consideration. This system function can be used in conjunction with **Set Selection Append Flag** to either append or replace the QBE or filter selection with the system function selection.

See Also:

- [Set Data Dictionary Item Overrides](#)
- [Set Selection](#)

Set QBE Column Compare Style

When the application performs a QBE search, this system function enables the application to set which comparison operator to use. For example, if you want to set the QBE for a date column to be > January 1, 2005, you would use QC WorkDate = "010105" and Set QBE Column Compare Style(FC Grid, GC WorkDate, <Greater Than>).

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column on which to set the QBE comparison operator. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Compare Style

Input, required. The comparison operator to use. Set the parameter to <Equal To>, <Not Equal To>, <Greater Than>, <Less Than>, <Greater Than or Equal To>, or <Less Than or Equal To>.

Set Selection

Use this system function to create the WHERE clause of a SQL search statement for use in a succeeding fetch.

Parameters

Grid

Input, required. The grid FC to affect.

Table

Input, required. The source from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Alias

Input, required. The column of the table from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Comparison Type

Input, required. The comparison operator to use. Set the parameter to the comparison operator to use (<Equal To>, <Not Equal To>, and so forth).

Comparison Value

Input, required. The comparison value to use. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list.

And/Or

Input, required. Indicates whether to append another WHERE clause to the SQL statement and how the new statement relates to the one before it (AND or OR). Set the parameter to <And>, <Or> or <None>.

Additional Notes

In general, this system function should not be used in cases where the user can affect the select. For example, QBE and filter fields are ways the user can affect the select. If the user can affect the select, then the user's select information should always be taken into consideration. This system function can be used in conjunction with **Set Selection Append Flag** to either append or replace the QBE or filter selection with the system function selection.

See Also:

- [Clear Selection](#)
- [Set Lower Limit](#)
- [Set Sequencing](#)
- [Set Selection Append Flag](#)

Set Selection Group

The Set Selection Group system function enables you to group together fetching criteria; this is the only characteristic of this system function that makes it different from the Set Selection system function. If you are not familiar with the Set Selection system function, please review the Grid Control System Functions section in the Form Design Aid guide.

The following three examples describe how Set Selection and Set Selection Group perform, and illustrate the differences in their functionality.

The result is that all employee records with the name ABC and all customer records are fetched.

Examples

Example 1 Set Selection

A form has a filter field called Alpha Name (ALPH), which is associated with the Alpha Name column in the F0101 table. In addition, there are two Set Selection system functions under the Find Button Clicked event. These two system functions are as follows:

- Set Selection (FC Grid,F0101,AT1,<equal>,'E',<AND>)
- Set Selection (FC Grid,F0101,AT1,<equal>,'C',<OR>)

AT1 is the Search Type column in table F0101. Value 'E' is for Employee and value 'C' is for Customer.

When a user enters ABC in the filter field and then clicks the Find button, the following conceptual WHERE clause is created:

WHERE ALPH=ABC AND AT1=E OR AT1=C

Example 2 Set Selection Group

A form has a filter field called Alpha Name (ALPH), which is associated with the Alpha Name column in the F0101 table. In addition, there are two Set Selection Group system functions under the Find Button Clicked event. These two system functions are as follows:

- Set Selection Group (FC Grid,F0101,AT1,<equal>,'E',<AND>)
- Set Selection Group (FC Grid, F0101,AT1,<equal>,'C',<OR>)

AT1 is the Search Type column in the F0101 table. Value 'E' is for Employee and value 'C' is for Customer.

When a user enters ABC in the filter field and then clicks the Find button, the following conceptual WHERE clause is created:

WHERE ALPH=ABC AND (AT1=E OR AT1=C)

The result is that all employee and all customer records with the name ABC are fetched.

Example 3 Set Selection and Set Selection Group

A form has a filter field called Alpha Name (ALPH), which is associated with the Alpha Name column in the F0101 table. In addition, there are two Set Selection system

functions and two Set Selection Group system functions under the Find Button Clicked event. These four system functions are as follows:

- Set Selection (FC Grid,F0101,AN8,<greater than>,'100',<AND>)
- Set Selection Group (FC Grid,F0101,AT1,<equal>,'E',<AND>)
- Set Selection (FC Grid,F0101,AN8,<less than>,'200',<AND>)
- Set Selection Group (FC Grid,F0101,AT1,<equal>,'C',<OR>)

AN8 is the Address Number column in the F0101 table; AT1 is the Search Type column in the F0101 table. Value 'E' is for Employee and value 'C' is for Customer.

When a user enters ABC in the filter field and then clicks the Find button, the following conceptual WHERE clause is created:

WHERE ALPH=ABC AND AN8>100 AND AN8<200 AND (AT1=E OR AT1=C)

The result is that all employee and all customer records with the name ABC and address number in the range of 100 to 200 are fetched.

Although shown in this example, the usage of mixed Set Selection and Set Selection Group is not a good practice. The purpose of this example is to show the runtime behavior - Set Selection Group system functions are grouped and appended to the WHERE clause at the end.

Set Selection Append Flag

When building a SQL statement programmatically, use this system function to indicate whether the returned value should replace the existing data or be appended to it.

Parameters

Grid

Input, required. The grid FC to affect.

Append?

Input, required. Indicates whether to append or replace. Set the parameter to append (<Yes>) or replace (<No>).

See Also: [Set Selection](#)

Set Sequencing

Use this system function to create the ORDER BY clause of a SQL search statement for use in a succeeding fetch.

Note: Use of this system function implies that grid formats that include sequencing will *not* be used in the SQL.

Parameters

Grid

Input, required. The grid FC to affect.

Table

Input, required. The source from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Alias

Input, required. The column of the table from which to acquire comparison data. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Sort Order

Input, required. Whether to sort in ascending or descending order. Set the parameter to <Ascending> or <Descending>.

Additional Notes

In general, this system function should not be used in cases where the user can affect the select. For example, QBE and filter fields are ways the user can affect the select. If the user can affect the select, then the user's select information should always be taken into consideration.

See Also: ■[Clear Sequencing](#)

- [Clear Sequencing](#)
- [Set Selection](#)
- [Set Selection Append Flag](#)

Show Grid Column

Use this function to make a hidden column visible. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column to show. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

See Also: [Hide Grid Column](#)

Show Grid Row

Use this function to reveal and display a hidden row. This system function affects the model version of the grid, and it is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to show. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or applicable object from the object list.

See Also: [Hide Grid Row](#)

Suppress Grid Line

Use this function to prevent a row from becoming part of the grid. This system function has an effect only when called from the **Grid Record is Fetched** event.

Parameter

Grid

Input, required. The grid FC to affect.

Update Grid Buffer Row

Use this system function to update a row from the grid buffer (GB) into the grid control. This system function affects the model version of the grid and is one-based.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to affect. Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or applicable object from the object list.

Selectable?

Input, required. An indicator of whether the Select button processes for the inserted row. Set the parameter to <Yes> or <No>.

Protected?

Input, required. An indicator of whether the user can edit the inserted row. Set the parameter to <Yes> or <No>.

Updateable?

Input, required. An indicator of whether runtime will attempt to update or insert the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable?

Input, required. An indicator of whether the user can delete the inserted row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the insert. Set the parameter to <Yes> or <No>.

See Also: [Insert Grid Buffer Row](#)

Grid

Input, required. The grid FC to affect.

Row

Input, required. The relative row to affect. Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, or applicable object from the object list.

Selectable?

Input, required. An indicator of whether the Select button processes for the inserted row. Set the parameter to <Yes> or <No>.

Protected?

Input, required. An indicator of whether the user can edit the inserted row. Set the parameter to <Yes> or <No>.

Updateable?

Input, required. An indicator of whether runtime will attempt to update or insert the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable?

Input, required. An indicator of whether the user can delete the inserted row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the insert. Set the parameter to <Yes> or <No>.

See Also: [Insert Grid Buffer Row](#)

Was Grid Cell Value Entered

This system function returns a nonzero value if a specific grid cell has been changed since last time this system function is called. Row is taken from the current context.

Parameters

Grid

Input, required. The grid FC to affect.

Column

Input, required. The column containing the grid cell to check. Set the parameter to an applicable object (GC grid column) from the object list or <All Columns>.

Return To

Input, required. The object to which to assign the return value that designates whether the cell or node has changed. Set the parameter to an applicable object from the object list.

This system function returns one of these values to the object identified by Row:

0

The cell has not changed since the last check.

1

The cell has changed since the last check.

Was Grid Cell Value Entered

22

Understanding Hot Keys

Hot keys are key stroke combinations, such as CTRL+ALT+V to save, that users can use instead of a mouse click to perform a **Button Clicked** event. In a JD Edwards EnterpriseOne application, most hot keys are reserved for system use.

This chapter contains the following topics:

- [Section 22.1, "System-Defined Push Button Hot Keys"](#)
- [Section 22.2, "System-Defined Toolbar Button Hot Keys"](#)
- [Section 22.3, "Application-Defined Hot Keys"](#)
- [Section 22.4, "Defining a Hot Key in Your Application"](#)

22.1 System-Defined Push Button Hot Keys

There is a set of push button hot keys that are created by the system when you use any of certain predefined text strings on a push button.

See "Understanding Hot Keys" in the *JD Edwards EnterpriseOne Tools Form Design Aid Guide*.

22.2 System-Defined Toolbar Button Hot Keys

There is another set of push button hot keys that are created by the system in JD Edwards EnterpriseOne forms.

Click on the 'i' button on any open JD Edwards EntperiseOne form to open the information dialog. Then, click on Display list of all hotkeys to obtain the list of hotkeys supported by the software.

22.3 Application-Defined Hot Keys

The hot keys that you can define in your applications are:

- CTRL+SHIFT+B
- CTRL+SHIFT+D
- CTRL+SHIFT+L
- CTRL+SHIFT+M
- CTRL+SHIFT+N
- CTRL+SHIFT+P
- CTRL+SHIFT+Q

- CTRL+SHIFT+U
- CTRL+SHIFT+X
- CTRL+SHIFT+Y

22.4 Defining a Hot Key in Your Application

To define a hot key, insert an ampersand before the letter that you want to set as the hot key in the Title property for the button that will have the hot key. You can set the definition from either the properties dialog or from the properties panel. The letter you define is then underlined in your application. Hot keys are governed by these rules:

- You can define only one hot key for a button.
- You cannot use the same hot key on more than one button in the same form or subform.
- You cannot use a reserved hot key (any that is not in the list of application-defined hot keys).
- You cannot insert more than one ampersand in the same button title.
- You must check the Hot Key property for an exit definition only. Hot keys are enabled by default for push buttons.

When you use the properties dialog to define the hot key and violate one of these rules, the system displays an error message and does not save the definition. However, when you use the properties panel to define the hot key, and violate a rule, no error message box is displayed. FDA ignores your definition (it is not saved).

23

Understanding Image Controls

This chapter contains the following topics:

- [Section 23.1, "Image Controls"](#)
- [Section 23.2, "Image Control Design-Time Considerations"](#)

23.1 Image Controls

You can use a bitmap control to create a control that looks like a picture or other artwork. You can then attach event rule logic to the control. For example, you can attach logic to the **Button Clicked** event on the control so that when a user clicks the control, the application automatically links to a different form. **Button Clicked** is the only event that can fire on an image control.

You also can use the bitmap control for an animated gif instead of a bitmap. An animated gif is particularly useful for Java and HTML applications. The animated gif is animated in Java and HTML applications; in Windows applications, however, it does not appear animated and only the first image of the animated gif file appears.

23.2 Image Control Design-Time Considerations

This list discusses how to use the design-time properties that are particularly useful for image controls:

- Clickable

If you want to make the image control act as a button (that is, fire the **Button Clicked** event when the user clicks it), enable this option. The option does not automate the image, however. Hence, the user receives no feedback when he or she clicks the control unless you add logic on the **Button Clicked** event to that end.

- Maintain Aspect Ratio

Select this option to maintain the height-to-width ratio of the image when you resize it.

- Prevent Resizing

Select this option to prevent your accidentally resizing the image during design-time.

- Tool Tip

To display text when the user hovers over the control, enter the text you want to show in this property field.

24

Understanding Media Object Controls

This chapter contains the following topics:

- [Section 24.1, "Media Object Controls"](#)
- [Section 24.2, "Media Object Control Design-Time Considerations"](#)
- [Section 24.3, "Media Object System Functions"](#)

24.1 Media Object Controls

The *media object control* is a specialized control. Use this control to enable the user to enter text or attach objects. You can place this control on any form type except the message box.

A single media object control can contain multiple items. Furthermore, depending on how it is configured, a media object can also contain a variety of different types of items. The user can only work with one at a time, however.

You can use the media object control in a variety of ways. You can add images, shortcuts to other applications, and text. You can add multiple media objects to a form. You can add multiple text objects to a single media object. You also can add generic files or URLs. If you want to display a file that is available on the internet, you can attach the media object control to the form and create a link to the internet.

After you define the media object queue for the internet and include a valid HTTP address, you can use the **Start Web Browser** system function to open the control and display the internet file. For example, you might use this control when you need to verify the Web page for a shipping vendor so that you can track the status of shipments. You can look up the shipment directly within the media object control.

You also can use the media object control to display the employee queues to which messages are sent. Place a media object control next to the tree control on a parent/child form. Next, use event rules (ER) to establish a relationship between the tree side and the media object side. For example, if a message is highlighted in the tree, then the corresponding message text appears in the control.

24.2 Media Object Control Design-Time Considerations

You can control the type of media objects users can attach by selecting one or more of these options:

- Allow Image Items
- Allow OLE Items
- Allow RTF Text

- Allow Text Items

Additionally, you can opt to cause a new text item to be created every time the control is opened by selecting the New Text Item on Open option.

Media object system functions enforce media object security in the web client. When applications that have media object security applied to them are running , the system logs the security information for the system functions in the web client debug log file.

24.3 Media Object System Functions

This section discusses the system functions unique to media objects.

Access Media Object

Use this system function to access a media object. Most of the media object system functions require you to use this system function to access a media object before you can manipulate it in another way.

Parameters

Media Object Control

Input, required. The media object FC to affect.

GTName

Input, required. The media object data structure to use. Set the parameter to an alphanumeric constant (<Literal>), a user prompt (<Choose GTName>), or an applicable object from the object list.

Key String

Input, required. The key string of the media object to access. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Action

Input, required. The display mode for the media object. You can show the media object as a read-only object, or you can enable users to edit the media object. Set the parameter to <Edit> or <Display>.

Status

Output, required. This parameter is reserved for future use.

Active Item

Input, optional. The item to activate (that is, bring to the forefront of the control). A media object can have more than one item associated with it. If you want an item of a certain type to be displayed initially, specify it with this parameter. You can also choose to enable the user to select the initial display item type as well. Set the parameter to <First Image Item>, <First OLE Item>, <First Text Item>, or <Specify Item#>.

Manage Media object

Use this system function to access a media object.

Parameters

GTName

Input, required. The media object data structure to use. Set the parameter to an alphanumeric constant (<Literal>), a user prompt (<Choose GTName>), or an applicable object from the object list.

GT Key

Input, required. The key string of the media object to access. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Delete

Input optional. If user wants the Delete option for the Media Objects uploaded, need to pass this parameter as<yes>.

Upload

Input, optional. If user wants the Upload option, pass this parameter as <yes>.

Download

Input, optional. If user wants the Download option for the Media Objects uploaded, pass this parameter as<yes>.

View

Input required. User has to pass the view as<yes> to view the Media Object files uploaded. If the User passed any of the Delete (OR) download parameters, the View parameter is taken as true since View is implicit by default.

Activate Item

Use this system function to activate (that is, bring forward) a specific item in a given media object. You must first access the media object itself before activating its items.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Item

Input, required. The ID of the item to activate. Set the parameter to <First Image Item>, <First OLE Item>, <First Text Item>, or <Specify Item#>. [Access Media Object](#)

Clear Characterization Cache

Use this system function to clear all values in the characterization cache that were set previously by the **Set Characterization Cache** system function. After this system function is called, adding media objects will not automatically generate a record in the F00166 table, which is where metadata about a media object is saved.

Parameters

Media Object Control

Input, required. The media object FC to affect.

GTName

Input, required. The media object data structure to clear. Set the parameter to an alphanumeric constant (<Literal>), a user prompt (<Choose GTName>), or an applicable object from the object list.[Disable Characterization Cache](#)
[Set Characterization Cache](#)

Delete Item

Use this system function to delete a specified item from a given media object. You must first access the media object itself before deleting its items.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Item

Input, required. The ID of the item to delete. Set the parameter to <First Image Item>, <First OLE Item>, <First Text Item>, or <Specify Item#>.Access Media ObjectActivate Item

Disable Characterization Cache

Use this system function to prevent runtime from changing the media object icon to represent the characterization of the media object.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Get OLE Item

This system function accepts a media object item sequence number and an OLE automation server interface as parameters and returns a handle to the OLE object with that sequence number in the media object record.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Item

Input, required. The ID of the item containing the OLE object to be fetched. Set the parameter to <First Image Item>, <First OLE Item>, <First Text Item>, or <Specify Item#>.

ObjectRef

Output, required. The object to which to return the ActiveX/COM interface variable. Set the parameter to an applicable object from the object list.

Returns

This system function returns the handle of the referenced OLE item to the object specified by ObjectRef. It also returns one of these two values:

NOERROR

Indicates that the system function succeeded.

E_FAIL

Indicates that the system function failed.[Insert OLE Object](#)[Insert URL](#)

Insert OLE Object

This system function adds an OLE object to a given media object.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Item Name

Output, required. The object to which to return the name of the item. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Item

Input, required. The object to which to return the next available ID. Set the parameter to an alphanumeric constant (<Literal>) or <Null>.

ObjectRef

Output, optional. Reserved for future functionality.

Returns

This system function returns the name of the inserted object, as well as the next available ID value to the objects indicated by Item Name and Item, respectively. It also returns one of these two values:

NOERROR

Indicates that the system function succeeded.

E_FAIL

Indicates that the system function failed.[Get OLE Item](#)

Insert Text

Use this system function to add text object to a given media object.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Text ID

Input, required. The ID to assign to the item being inserted. Set the parameter to <Default Text Object> or an alphanumeric constant (<Literal>).

Text

Input (EVDT_STRING), required. The actual text to be inserted. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Insert URL

Use this system function to place an HTML page in a media object. You must pass in the media object queue name where the HTML file resides and the name of the file.

Parameters

Media Object Control

Input, required. The media object FC to affect.

MO Queue Name

Input, required. The media object queue in which HTML files reside. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

HTML File Name

Input, required. The name of the HTML file to be attached. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Returns

This system function returns one of these two values:

NOERROR

Indicates that the system function succeeded.

E_FAIL

Indicates that the system function failed.[Get OLE ItemInsert OLE Object](#)

Hide the Viewer Icon Panel

This system function hides the viewer icon panel on a given media object control. When the panel is hidden, users cannot select particular media objects to view, although they can interact with any media object displayed in the main portion of the control. You can use the **Activate Item** system function to display different items in the media object programmatically.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Hide Icon Panel?

Input, required. Indicates whether to hide the icon panel. Set the parameter to **<Yes>** or **<No>**.

Lock the Viewer Splitter Bar

This system function exists for backwards compatibility. Do not use it.

Set Characterization Cache

Use this system function to set data in the characterization cache. This cache is saved in the F00166 table and is used as metadata.

Parameters

Media Object Control

Input, required. The media object FC to affect.

GTName

Input, required. The media object data structure to use. Set the parameter to an alphanumeric constant (<Literal>), a user prompt (<Choose GTName>), or an applicable object from the object list.

Categories

Input, required. The data values to use as metadata. Double-click <Define Characterization> to set the values.

Set Cursor Position

This system function sets the cursor in a text type media object item either at the beginning of the text (Home position).

Note: This system function has been marked for deprecation in a future release; therefore, avoid its use.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Text ID

Input, required. The ID fo the text item that you want to affect. You can select a specific item, or you can choose to affect the default text item if you have configured the control at runtime so that a default text item launches each time the user opens the control. Set the parameter to *<Default Text Object>* or an alphanumeric constant (*<Literal>*).

Position

Input, required. The position at which to place the cursor. Set the parameter to *<HOME>*.

Set Grid Text Indicator

Use this system function to determine whether a media object is associated with a given row record. If so, runtime can display a paperclip icon in its row header as a signal to the user that the media object exists.

Parameters

Grid

Input, required. The grid FC to affect.

Row

Input, required. The row to check. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, or an applicable object from the object list.

Set Indicator?

Input, required. Indicates whether runtime should display the paperclip icon for the user if a media object is associated with the row. Set the parameter to <Yes> or <No>.

Set Text Color

This system function enables you to set the text color for a text-type item in a given media object control.

Parameters

Media Object Control

Input, required. The media object FC to affect.

Text ID

Input, required. The ID of the text item you want to affect. You can select a specific item, or you can choose to affect the default text item if you have configured the control at runtime so that a default text item launches each time the user opens the control. Set the parameter to <Default Text Object> or an alphanumeric constant (<Literal>).

Color

Input, required. The color to be set. Set the parameter to a color from the color palette (<Pick Color>) or the default color value (<Reset Color>).

25

Understanding Parent Child Controls

This chapter contains the following topics:

- [Section 25.1, "Parent Child Controls"](#)
- [Section 25.2, "Tree Nodes"](#)
- [Section 25.3, "Lean Manufacturing"](#)
- [Section 25.4, "Parent Child Control Design-Time Considerations"](#)
- [Section 25.5, "Parent Child Control Events"](#)
- [Section 25.6, "Parent Child Control System Functions"](#)

25.1 Parent Child Controls

The parent child control is a composite control with a tree on the left and a grid on the right. The control is used to provide a hierarchical view of the business data. Users can resize the tree part of the control during runtime. The tree and grid portions use the same business view, if one is attached. If an editable parent child control links to a business view and the user has edited the content, the change is saved by runtime when the form is saved.

At most, a form can contain a single parent child control. Additionally, a form cannot contain both a parent child control and a grid control. You cannot place a parent child control on a tab page; although you can nest a reusable subform containing a parent child control on a tab page.

You can add data dictionary (DD) items to the grid. Because of its similarity to a grid control, many of the properties for parent child controls are the same.

After you configure the control, you set up either a parent/child relationship or use event rules (ER) to load child nodes to the tree. The method that you use depends on whether the table has an inherent parent/child relationship.

You can use ER and system functions to customize the way in which the parent child control functions. For example, you can change the top node of a tree or change the node that appears as the first child on a tree.

To increase performance during runtime, parent child controls can make use of page-at-a-time processing. Page-at-a-time processing ensures that each fetch fetches only one page of data. Page-at-a-time processing is the default mode for all parent/child forms. During page-at-a-time processing in standard mode, the page size is the number of nodes that can fit in the current view. When the Find process begins, only one page of first-level nodes is fetched from the table and inserted in the tree. When the user scrolls down, a new page of data is fetched.

Page-at-a-time processing for the expand-all style of parent/child forms is similar to that for standard mode. When the Find process starts, one page of first-level nodes is fetched from the table and inserted into the tree. Then, all of the first-level nodes are expanded. Each expansion fetches only one page of data. Because the tree expands exponentially in expand-all mode, a deep tree might affect performance.

See Also:

- [Understanding Grid Controls](#).

25.2 Tree Nodes

When the parent and child nodes come from different tables or are of different data types, the parent/child relationship is not automatically set up. In this case, the runtime engine does not automatically fetch the child database records because it does not know the table from which to retrieve them.

If possible, use the runtime engine to load the initial set of parent nodes to the tree for you. You do this by using the based-on view, which is a view over the table for the uppermost node. You can use a parent filter in the control, and the runtime engine loads the first level nodes to the tree. If you cannot do this, you must insert the first-level nodes yourself. To do this, you typically use table I/O on the **Button is Clicked** event of the Find button. You use the same methods that you use to insert child nodes. Use a **Suppress Find** system function to stop the runtime engine from attempting to load any nodes.

Whenever a node is expanded, the system function, **Suppress Fetch on Node Expand**, is called from the event, **Tree Node Is Expanded**. This function tells the runtime engine not to do any fetches because ER will handle the loading of the child nodes. **Tree Node Is Expanding** is the main event of the application. This event occurs when the tree node is expanding (such as when the plus next to a child node is expanded for the first time). You place ER on this event to read the next records to be loaded to the tree as children of the expanded node. You can use table I/O or business functions to retrieve these records. Often the children come from different tables, based on the type of parent node that is expanded. If possible, you should perform a **SELECT** and then use the **FETCH NEXT** command to retrieve records in a **DO WHILE** loop. The grid buffer (GB) runtime data structure is populated with data from the records read in the loop, and then an **Insert Grid Buffer Row** system function is called. This parent/child system function is different from the **Insert Grid Buffer Row** in the normal grid section. At this point, you also can set custom tree bitmaps using the **Set Tree Node Bitmap** system function.

25.3 Lean Manufacturing

Lean manufacturing (also known as *Psynch* or *Product synch*) is a special mode of the parent child control that was created to be used in applications designed to support lean manufacturing processes. It displays a parent child grid in a graphical format specific to lean manufacturing. In many respects, including runtime processing, the lean manufacturing control functions in the same way as the standard parent child control.

25.4 Parent Child Control Design-Time Considerations

This section discusses parent child control properties and power forms.

25.4.1 Parent Child Control Properties

Parent child controls are a specialized form of the grid control. Therefore, many of the design options that apply to grid controls also apply to parent child controls. This table details some key design-time settings specifically for the parent child control and their impact. All locations are relative to the parent child control Properties form:

Setting	Impact
Multiple Select	Select this option to permit the user to select multiple lines to affect with a single operation. If cleared (the default), users cannot affect multiple nodes for operations such as cut-and-paste.
Disable Drag & Drop (Move), Disable Copy, Disable Drag & Drop (Cut/Copy/Paste), Move Up and Down, Indent and Outdent	Select these options to enable the user to affect the contents and organization of the tree.
Location Indicator Feature	Select this option to provide a button that permits the user to toggle showing and hiding location indicators in the tree. The location indicator is a numerical representation of the position of a node in the tree.
Expand All/Collapse All	Select this option to provide a button for the user to expand or collapse the entire tree.

25.4.2 Parent Child Control and Power Forms

If you place the parent child control on a power edit form, an embedded subform on a power edit form, or an editable reusable subform, then the parent/child will be editable within these parameters:

- Both tree columns and the grid columns (except for parent child relationship columns) are editable unless they have been rendered read-only by application logic.
- Columns that define the parent/child key relationship are not editable.
- Runtime provides format and validation for grid cells and tree nodes, similar to the one provided for regular grid.
- The column events, **Column Is Exited**, **Column Is Exited and Changed In-Line**, and **Column Is Exited and Changed - Asynchronous**, are supported for the tree column and all grid columns.

The Trigger Parallel Event system function is available for the **Column Is Exited and Changed** and **Column Is Exited and Changed In-Line** events. This system function will enable a parallel event to run on a separate thread and will not interfere with existing Event Rules.

- The row events, **Row Is Exited**, **Row Is Exited and Changed In-Line**, **Row Is Exited and Changed - Asynchronous**, are supported for parent child controls.

The Trigger Parallel Event system function is available for the **Row Is Exited and Changed** and **Row Is Exited and Changed In-Line** events. This system function will enable a parallel event to run on a separate thread and will not interfere with existing Event Rules.

- The event sequence is the same as the grid event sequence.
- Form default buttons such as OK and Delete work for parent child controls as they do for grid controls.

At most, a power form or a subform can contain a single parent child control directly. However, a power form can contain multiple subforms, and each subform can contain a parent child control. Similarly, while you cannot place a parent child control on a tab page, you can create a subform as a tab page and that subform can contain a parent child control.

25.4.3 Lean Manufacturing Properties

Place the parent child control in PSYNC mode. This enables the Product Sync Mapping property for grid columns in the control. For every item in the property, you must place a corresponding column in the grid and map it to the property.

Grid columns in the parent child control in PSYNC mode have the Product Synch Mapping property. This list summarizes its parameters:

- Attach Path to Segment
Parameter 2 no longer permits the <Currently Selected Row> special value.
- Change Row Selection
Parameter 3 special value is <Unselected> (as opposed to <Deselected>).
- Set Action
Parameter 2 has no special values. Parameter 3 special values are <Enabled (1)> and <Disabled (0)>.

25.5 Parent Child Control Events

These events can fire on the parent child control during runtime:

- Delete Grid Rec Verify - Before
- Delete Grid Rec Verify - After
- Delete Grid Rec from DB-Before
- Delete Grid Rec from DB-After
- Double Click on Row Header
- Set Focus on Grid
- Kill Focus on Grid
- Row is Entered
- Get Custom Tree Node
- Tree Node Bitmap is Clicked
- Node Outdent Verify Before
- Tree Node is Outdented
- Node Indent Verify Before
- Tree Node is Indented
- Node Move Down Verify Before
- Node is Moved Down
- Node Move Up Verify Before
- Node is Moved Up

- Tree-Node Level Changed
- Tree - Begin Drag/Drop/Copy
- Tree - Drag Over Node
- Tree - Cancel Drag Drop/Paste
- Tree - End Drag Drop/Paste
- Kill Focus on Control
- Set Focus on Control
- Tree Node Is Collapsing
- Tree Node Is Expanding
- Tree Node Selection Change
- Tree Node Is Deleted
- Row is Exited

Many of these events are unique to parent child controls and can be categorized for discussion:

- Selecting tree nodes.
- Performing drag-and-drop or copy/cut/paste.
- Expanding and collapsing nodes.
- Clicking bitmaps.

25.5.1 Selecting Tree Nodes

The **Tree Node Selection Changed** event runs every time a user selects a tree node, either by clicking it once with the mouse or by moving an arrow up and down the nodes. You can place ER that needs to run when a mode is selected on the **Tree Node Selection Changed** event. The Work Center application uses this feature to load the media object that appears next to the tree with the message information for each node. You can also use this event when you need to protect controls or exits, based on the kind of node that the user selects.

25.5.2 Performing Drag-and-Drop or Copy/Cut/Paste

If you enable the functionality, users can affect the tree structure by adding, moving, and deleting nodes. In Windows environments, users accomplish these tasks with drag-and-drop functions. In Web environments, users accomplish these tasks with copy/cut/paste.

When you place the control on a form initially, all three options are enabled:

- Cut (drag-and-drop move).
- Copy.
- Cut-copy-paste (drag-and-drop).

You can disable these options at design time. When an option is enabled and a user attempts the operation, the cursor indicates that the function is not permitted, and none of the associated events execute. You can control operations to the database using these events:

- **Tree - Begin Drag/Cut/Copy**

If you enable any of the three operations, the **Begin Drag** event fires, and the drag mode is set to Move for the first two operations (cut or copy). The mode is set to Copy for the third operation (cut-copy-paste).

- **Tree - Drag Over Node**

You can attach ER to this event to verify that the node on which the dragged record is about to be dropped is a valid situation. If it is not, you can use a system function to change the cursor to a No Drop cursor to indicate that dropping the record there is not permitted. If the cursor is not the No Drop cursor, when the record is dropped the event, **Tree - End Drag Drop/Paste**, runs. The same is true of cut-and-paste or copy-and-paste operations.

- **Tree - End Drag Drop/Paste**

You can attach ER to this event to update or insert information that has been moved or copied using the drag. Be aware of the effect of using **Insert Grid Buffer Row** in the **Tree -End Drag Drop/Paste** event, as well as deleting the grid row dragged if the mode is a move.

- **Tree - Cancel Drag Drop/Paste**

No matter the operation, if the user cancels, then this event occurs.

In addition to drag-and-drop or copy/cut/paste, users (and applications) can simply move nodes up and down or change their hierarchical level (indent and outdent). These events are provided in connection with these activities:

- **Node Move Up Verify Before**

When an attempt to move a node up occurs, runtime fires this event. If the system function, **Suppress Node Move Up/Down** is called within this event, then move up is canceled. Otherwise, runtime moves the node upwards one position and then fires the event, **Node is Moved Up**.

- **Node Is Moved Up**

When a node has been moved up by one position (if permitted on the event, **Node Move Up Verify Before**), then the event, **Node is Moved Up**, is fired for that node. Its child nodes are still parented to the moved node. You cannot move a node that is the first node on its level up.

- **Node Move Down Verify Before**

When an attempt to move a node down occurs, runtime fires this event. If the system function, **Suppress Node Move Up/Down** is called within this event, then move down is canceled. Otherwise, runtime moves the node downwards one position and then fires the event, **Node is Moved Down**.

- **Node Is Moved Down**

When a node has been moved down by one position (if permitted on the event, **Node Move Down Verify Before**), then the event, **Node is Moved Down**, is fired for that node. Its child nodes are still parented to the moved node. You cannot move a node that is the last node on its level down.

- **Node Indent Verify Before**

When an attempt to indent a node occurs, runtime fires this event. If the system function, **Suppress Node Indent/Outdent**, is called within this event, then indent is canceled. Otherwise, runtime demotes the node to be a child of its previous sibling (the demoted node remains the parent of its children). Then the event, **Tree Node is Indented**, is fired.

- **Tree Node is Indented**

When a node has been indented (if permitted on the event, **Node Indent Verify Before**), then the event, **Tree Node is Indented**, is fired for that node. Its child nodes are still parented to the indented node. The application must update the node's internal data to reflect its new parent. You cannot indent a node that is the first node on its level.

- **Node Outdent Verify Before**

When an attempt to outdent a node occurs, runtime fires this event. If the system function, **SUPPRESS NODE INDENT/OUTDENT**, is called within this event, then outdent is canceled. Otherwise, runtime promotes the node to be a parent of the sibling beneath it. The original parent becomes the sibling of the outdented node. If the outdented node is the first child, then the original parent loses all of its children to the outdented node and becomes a leaf node. The child nodes of the outdented nodes are otherwise unaffected. Then the event, **Tree Node is Outdented**, is fired.

- **Tree Node is Outdented**

When a node has been outdented (if permitted on the event, **Node Outdent Verify Before**), then the event, **Tree Node is Outdented**, is fired for that node. The application must update the node's internal data to reflect its new parent.

- **Tree Node Bitmap is Clicked**

Under most conditions, each node in the tree has a bitmap next to it for display purposes only. You can use **Set Tree Node Clickable Bitmap** to provide an additional bitmap which is clickable; that is, clicking the bitmap causes the event, **Tree Node Bitmap is Clicked**, to fire.

25.5.3 Expanding and Collapsing Nodes

ER exists to signal when a node has been expanded or collapsed. These events occur whether the change in the node's status is due to the user or the application:

- **Tree Node Is Expanding**

This event occurs when a node is expanded for the first time after control initialization.

- **Tree Node Is Collapsing**

This event occurs when a node is collapsed after control initialization.

25.5.3.1 Example: Using the Tree Node is Expanded Event

In this example, ER is attached to an application on the **Tree Node Is Expanding** event:

```
Suppress Fetch on Node Expand(FC parent/child)
//
// Here are the variables to get out of the account and the business unit loop
// being initialized.
VA frm_OutOfLoop = "0"
VA frm_ExistAcctLoop = "0"
//
// If Loop looking at the GC Business Unit Field.
If GC BusinessUnit is equal to <Blank>
//
// Select the F0006 differently if there is one company or all companies
//
```

```
F0006.Open
If VA frm_AllCompanies is equal to "1"
VA frm_CurCompany = GC Co
F0006.Select
Else
VA frm_CurCompany = BC Company
F0006.Select
End If
//
// While Loop which fetches the business units for a specific company.
// If the company changes we get out of the loop.
While VA frm_OutOfLoop is equal to <Zero>
// Fetch the records from the F0006 Table.
F0006.FetchNext
GB BusinessUnit = GB Companies
If SV File_IO_Status      is equal to CO SUCCESS
GB Co = BC Company
VA frm_PreCompany = BC Company
VA frm_ConcateBuDesc = " "
VA frm_ConcateBuDesc = concat([VA frm_ConcateBuDesc],[VA frm_NameOfBU])
GB CompanyStructure = concat([GB Companies],[VA frm_ConcateBuDesc])
GB Companies = concat([GB Companies],[VA frm_ConcateBuDesc])
//
// Tells the Level of the Tree Structure.
GB LevelOfTreeInt = "1"
//
Insert Grid Buffer Row(FC parent/child, <After Last Row>, <Yes>, <No>,
<No>, <No>, <No>, <Yes>)
Set Tree Node Bitmap(FC parent/child, <Last Grid Row>, BussUnit.bmp, <Yes>)
//
//
If VA frm_PreCompany is not equal to VA frm_CurCompany
VA frm_OutOfLoop = "1"
End If
Else
VA frm_OutOfLoop = "1"
End If
End While
F0006.Close
Else
// Loop thru the F0901 to get the corresponding accounts.
//
VA frm_CURBU = GC BusinessUnit
F0901.Open
F0901.Select
If SV File_IO_Status      is equal to CO ERROR
//
End If
//
// While Loop to pick up the accounts till the Fetch Fails.
While VA frm_ExistAcctLoop is equal to <Zero>
F0901.FetchNext
GB Sub = VA frm_DBSub
GB ObjAcct = VA frm_DBOBJ
GB Name = VA frm_AcctDesc
GB BusinessUnit = VA frm_CURBU
VA frm_AcctDesc = concat([VA frm_BLANKS],[VA frm_AcctDesc])
GB AccountID = VA frm_AIDF0901
GC AccountID = VA frm_AIDF0901
If SV File_IO_Status      is equal to CO ERROR
```

```

VA frm_ExistAcctLoop = "1"
Else
  GC Companies = " "
  GB Companies = " "
  Business Unit, Object, Subsidiary Merge
  GB Companies = concat([VA frm_DBANI], [VA frm_AcctDesc])
  GB CompanyStructure = concat([VA frm_DBANI], [VA frm_AcctDesc])
  GC Companies = VA frm_AIDF0901
  //
  // Tells the level of the Tree Structure '2'
  GB LevelOfTreeInt = "2"
  //
  Insert Grid Buffer Row(FC parent/child, <After Last Row>, <Yes>, <No>, <No>,
    <No>, <No>, <No>)
  Set Tree Node Bitmap(FC parent/child, <Last Grid Row>, accounts.bmp, <Yes>
  End If
  End While
  End If
  FC BUFrom = " "

```

25.5.4 Clicking Bitmaps

With the system function, **Set Tree Node Clickable Bitmap**, you can display a bitmap next to a node which fires the event, **Tree Node Bitmap is Clicked**, when the user clicks the bitmap.

25.6 Parent Child Control System Functions

This section discusses the system functions unique to the parent child control.

In general, you can use system functions to perform these types of tasks in the parent child control:

- Affect user interactions with the control.
- Acquire information about a node.
- Load data into the control.
- Format data in the control.

Users have the ability to change the tree's hierarchical structure by moving nodes up and down. Users can also increase the indent or outdent of a node as well. To prevent vertical changes to the tree structure, use **Suppress Node Move Up/Down**. To prevent horizontal changes, use **Suppress Node Indent/Outdent**. Typically, you will call these system functions inside the events, **Node Indent Verify Before**, **Node Outdent Verify Before**, **Node Move Up Verify Before**, or **Node Move Down Verify Before**, to prevent the user from performing the action (move up/down or indent/outdent).

Under most conditions, each node in the tree has a bitmap next to it for display purposes only. You can use **Set Tree Node Clickable Bitmap** to provide an additional bitmap which is clickable; that is, clicking the bitmap causes the event, **Tree Node Bitmap is Clicked**, to fire.

You can use system functions to acquire this information about any node in the tree:

- Given its location, the node ID (**Get Node ID**).
- Given its ID, the node row number (**Get Row Number**).
- Given its ID, the node parent, siblings, and child IDs (**Get Related Node ID**).

When application logic is used to load the tree instead of the runtime logic, the application must often insert a tree node at a specific location based on data column values. To achieve this, at design time, select a column to be the node ID column. An application can specify a column as the node ID column. The value in this column will be used as a unique identifier for that row. The node ID column can be any data type, but must be unique. Use the system functions, **Insert Grid Buffer Row By Node ID** and **Get Related Node ID**, to specify a node ID column. Node ID column and these system functions work together: Node ID column needs to exist for these system functions to work. If none of these system functions is called, you do not need to specify a node ID column.

After you specify a node ID column, use the **Insert Grid Buffer Row By Node ID** system function to insert a row at any location, based on its node ID value and Insert As parameter. Use this method to permit an application to insert tree nodes at any position of the tree. Calling the **Insert Grid Buffer Row By Node ID** system function is the only way to add an entry row in the parent child control.

The parent child control provides some default bitmaps for tree nodes that are displayed to indicate the current status of that node (expanded, collapsed, and so forth). You can substitute your own bitmap to display for a node, however, with **Set Tree Node Bitmap**.

For each node, the parent child control can automatically generate and update hierarchical numbers called location indicators. If you enable the feature at design time, then users will see the location indicators button at runtime. By default, the numbering schema is 1, 1.1, 1.2, 1.3, 2, 2.1, and so on. You cannot modify the location indicator in any other way.

To insert an entry row in an empty parent child control, you must first call the system function, **Set Root Node ID**, to assign an ID to the hidden root node. By default, the root node in an empty tree is hidden, and therefore has no ID. After assigning an ID to the root node, then call the system function, **Insert Grid Buffer Row By Node ID**, and pass in the root node ID. If the control does not have a node ID column, runtime produces an error.

Add Action

Use this system function to add a context-based action to the parent child control in Product Sync mode.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Action Name

Input, required. The name of the action to add. Set the parameter to <Separator> or an applicable object from the object list.

Action ID

Input, required. The ID of the action to add. Set the parameter to <Separator> or an applicable object from the object list.

Parent Action ID

Input, required. Reserved for future functionality. Set to <Root Action>.

Icon

Input, required. The icon to display. Set the parameter to <Choose Action Bitmap>, <Default>, or <Separator>.

Additional Notes

If Action Name, Action ID, or Icon are specified as <Separator> then the entire action is a separator, and the values of the other parameters are to be ignored.

Attach Path To Segment

Use this system function to add a PSYNC path to a specific PSYNC segment or to all the segments in the PSYNC graph. It recursively adds all of the segments of a path to a given segment, facilitating reuse.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Product Synch ID

Input, required. The PSYNCH node to add the path. Set the parameter to <Currently Selected Row> or an applicable object from the object list.

Path ID

Input, required. The path to add to the segment. Set the parameter to an applicable object from the object list.

Error Code

Input, required. The object to which to assign the return value that indicates whether the function executed without error. Set the parameter to an applicable object from the object list.

Returns

This system function returns one of these values to the object identified by Error Code:

0

No error

1

Recursive

2

Other error

Change Row Selection

Use this system function to select or deselect a row in the grid or parent child control in Product Synch mode.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Row

Input, required. The row to affect. Set the parameter to **<All Rows>**, **<Currently Selected Row>**, or applicable object from the object list.

Select State

Input (EVDT_INT), required. The mode (such as Selected or Deselected) to which to set the row. Set the parameter to **<Selected>** (1), **<Deselected>** (0), or an applicable object from the object list.

Clear Grid Buffer

Use this system function to clear the GB manually.

Parameter

Parent/Child - Grid

Input, required. The parent child FC to affect.
[Copy Grid Row To Grid Buffer](#)
[Update Grid Buffer Row](#)

Clear Grid Cell Error

Use this system function to clear an error on a cell, a tree node, a row, a column or the entire parent child control. Also, this system function does not take hidden rows into account; in other words, it affects the user grid instead of the model grid.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Row

Input, required. The row in which to clear the error. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column in which to clear the error. Set to <All Columns>.

Clear QBE Column

Use this system function to clear text from the QBE columns in the control.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Column

Input, required. The QBE column to clear. Set to <All Columns>.

Contact Tree Node

Use this system function to collapse a tree node (for example, so that none of its children are displayed).

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, required. The target node to affect. Set the parameter to <Currently Selected Node>, <Last Inserted Node>, or <Currently Expanding/Collapsing Node>.

Copy Grid Row To Grid Buffer

Use this system function to write all of the GC fields to GB. Depending on the conditions when this function is initiated, the GC can be either in memory or in the grid.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The row to copy. Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, <Currently Expanding/Collapsing Node>, or an applicable object from the object list.[Clear Grid Buffer](#)[Update Grid Buffer Row](#)

Delete All Actions

Use this system function to remove all context-based actions from the specified parent child control in Product Synch mode.

Parameter

Parent/Child

Input, required. The parent child FC to affect.

Delete All Tree Nodes

Use this system function to delete all of the tree nodes (except for the hidden root) from a tree.

Parameter

Parent/Child

Input, required. The parent child FC to affect.

Delete Grid Row

Use this function to delete a grid row from the model grid. That is, you can use this system function to affect hidden rows.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative location to insert the row. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <All Child Nodes Under the Current Node>, <Drag Node Row>, <Currently Selected Row>, or an applicable object from the object list.

Disable Grid

Use this system function to disable a cell, a tree node, a row, the column or the entire parent child control. This system function does not take hidden rows into account when deleting a row based on an absolute row number value.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Row

Input, required. The row to be disabled. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column to be disabled. Set to <All Columns>. [Update Grid Buffer Row](#)

Enable Grid

Use this system function to enable a grid cell, a tree node cell, a row, a column or the entire parent child control. This system function does not take hidden rows into account when deleting a row based on an absolute row number value.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Row

Input, required. The row to be enabled. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column to be enabled. Set to <All Columns>. [Disable Grid](#)

Expand Tree Node

This system function expands nodes at a given hierarchical level and lower and collapses the rest, all relative to a given node. Expanding a very large tree (for example, one with 1000 nodes or more) can cause a reduction in performance.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, required. The target node to expand. Set the parameter to <Currently Selected Node>, <Last Inserted Node>, or <Currently Expanding/Collapsing Node>.

Get Grid Row

Use this function to target a grid row for use in an upcoming function in place of the grid row that would be used ordinarily.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row to target. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Get Max Grid Rows

Use this system function to count the number of rows in the model grid; that is, it counts both hidden and visible rows.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row to target. Set the parameter to an applicable object from the object list.

Get Next Selected Row

Use this system function to get the next selected row from a starting position in Product Synch mode. This applies to the display grid and not the model grid.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Start Row

Input, required. The row under which to find the next selected row. Typically, you pass **<Before First Row>** on the first row and then the value of the current row on subsequent calls. Set the parameter to **<Before First Row>** or an applicable object from the object list.

Selected Row

Output, required. The object to which to assign the value indicating the next selected row. Set the parameter to an applicable object from the object list.

Returns

This system function returns a value to the object identified by Selected Row. If no rows are selected after the start row, then the system function returns a value of -1.

Get Node ID

This system function returns the ID of a node at a given location. You can use this system function only if the control has a node ID column.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Node Location

Input, required. The location of the node for which you want an ID. Set the parameter to <Root Node>, <Currently Selected Node>, <Last Inserted Node>, <Currently Expanding/Collapsing Node>, <Currently Deleted Node>, <Drag Over Node>, or an applicable object from the object list.

Node ID

Output, required. The object to which you want to assign the return value that designates the node ID. The value must be a variable or control with the same data type as the node ID column; a type mismatch will be checked by the runtime engine. Set the parameter to an applicable object from the object list.

Error Code

Output, required. The object to which you want to assign the return value that designates the error code for the transaction.

Returns

This system code returns two values. The node ID itself is returned to the object identified by Node ID. A value indicating whether the node was actually found is sent to the object identified by Error Code and can have one of two values:

0

The node ID was found successfully.

Nonzero value

The node ID was not found.[Get Related Node ID](#)[Insert Grid Buffer Row By Node ID](#)

Get Node Level

This system function returns the hierarchical level of a given node.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node ID

Output, required. The object to which you want to assign the return value that designates the node ID. The value must be a variable or control with the same data type as the node ID column; a type mismatch will be checked the runtime engine. Set the parameter to <Currently Selected Node>, <Drag Over Node>, <Currently Expanding/Collapsing Node>, or <Currently Deleted Node>.

Return to

Output, required. The object to which you want to assign the return value that designates the node level. Set the parameter to an applicable object from object list.

Returns

This system function returns the hierarchical level of a given node to the object identified by Return to. If this system function fails (for example, the node ID does not exists), then the system function returns a value of -1.

Get Related Node ID

This system function returns the ID of a node related to a reference node.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Current node ID

Input, required. The ID of node to use as a reference. Set the parameter to an applicable object from the object list.

Relation

Input, required. The relationship of the target node to the reference node. Set the parameter to **<Parent Node>**, **<First Child Node>**, **<Next Sibling Node>**, **<Previous Sibling Node>**.

Node ID

Output, required. The object to which you want to assign the return value that designates the node ID. The value must be a variable or control with the same data type as the node ID column; a type mismatch will be checked by the runtime engine. Set the parameter to an applicable object from the object list.

Error Code

Output, required. The object to which you want to assign the return value that designates the error code for the transaction. The value must be integer type variable or control. Set the parameter to an applicable object from the object list.

Returns

This system code returns two values. The node ID itself is returned to the object identified by Node ID. A value indicating whether the node was actually found is sent to the object identified by Error Code and can have one of two values:

0

The node ID was found successfully.

Nonzero value

The node ID was not found.

Get Row Number

This system function returns the row number of a given node. You can use this system function only if the control has a node ID column.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Node ID

Input, required. The ID of the target node. Set the parameter to an applicable object from the object list.

Row Number

Output, required. The object to which you want to assign the return value that designates the row number.

Returns

This system function returns the next selected row number to the object identified by Row Number. If this system function fails (for example, the node ID does not exist), row number is -1.[Get Node ID](#)

Get Selected Context Action

Use this system function to get the action ID of the action that triggered the **OnAction** event. You cannot use this system function in any other event.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Action ID

Output, required. The object to which to assign the value that indicates the action ID of the action that triggered the **OnAction** event. Set the parameter to an applicable object from the object list.

Returns

This system function returns the ID of the action that triggered the **OnAction** event to the object specified by Action ID.

Get Selected Grid Row Count

Use this function to count the number of grid rows in a given range of rows (that is, in the current selection).

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Output, required. The object to which to assign the return value that designates the number of rows. Set the parameter to an applicable object from the object list.

Returns

This system function returns the number of rows in the current selection to the object specified by Row.

Get Selected Grid Row Number

Use this function to get the row number for a selected row. Typically, you use this function only when you need to save the row as a variable for future processing.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Output, required. The object to which to assign the return value that designates the row number. Set the parameter to an applicable object from the object list.

Returns

This system function returns the row number for the selected row to the object specified by Row.

Get Tree Node Handle

Use this system function to return the handle of a target node.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, optional. The ID of the node for which to get the handle. Leave this parameter blank if you want to identify the node by its position (Index parameter). Set the parameter to <Currently Selected Node>, <Last Inserted Node>, <Root Node>, <Currently Expanding/Collapsing Node>, or <Currently Deleted Node>.

Index

Input, optional. The position of the node for which to get the handle. Leave this parameter blank if you want to identify the node by its condition, such as currently selected, last insert, and so forth (Node parameter). Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Handle

Output, required. The object to which to assign the return value that designates the handle of the target node (numeric). Set the parameter to an applicable object from the object list.

Returns

This system function returns the handle for a tree node to the object specified by Handle.

Hide Grid Column

Use this system function to hide the entire column.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Column

Input, required. The column to hide. Set this variable to <All Columns>.Get Node ID

Insert Grid Buffer Row

Use this system function to insert a row from the GB into the control. This system function does not take hidden rows into account.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative location where to insert the data. Set the parameter to an alphanumeric constant (<Literal>), <After Current Row>, <After Last Row>, <Under Currently Expanding Node>, <Under Drop Node>, <Under Drop Node as First Child>.

Selectable?

Input, required. An indicator of whether the user can select the inserted row. Set the parameter to <Yes> or <No>.

Protectable?

Input, required. An indicator of whether the user can edit the inserted row. Set the parameter to <Yes> or <No>.

Updateable?

Input, required. An indicator of whether runtime will attempt to update the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable?

Input, required. An indicator of whether the user can delete the inserted row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the insert. Set the parameter to <Yes> or <No>.

Expandable?

Input, required. An indicator of whether the user can expand the inserted row. Set the parameter to <Yes> or <No>. [Insert Grid Buffer Row By Node ID](#)

Insert Grid Buffer Row By Node ID

Use this system function to insert a row from the GB into a location relative to another tree node.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Base On Node ID

Input, required. The ID of the target node. Set the parameter to an applicable object from the object list.

Insert As

Input, required. The position to insert the GB row, relative to the target node. Set the parameter to insert the row immediately under the target node and make it a child of the target <First Child Node>, make the row a child of the target node and insert it at the bottom of the target's children <Last Child Node>, insert the row immediately after the target node <Next Sibling Node>, or insert the row immediately before the target node <Previous Sibling Node>.

Selectable

Input, required. An indicator of whether the user can select the inserted row. Set the parameter to <Yes> or <No>.

Protectable

Input, required. An indicator of whether the user can edit the inserted row. Set the parameter to <Yes> or <No>.

Updateable

Input, required. An indicator of whether runtime will attempt to update the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable

Input, required. An indicator of whether the user can delete the inserted row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the insert. Set the parameter to <Yes> or <No>.

Expandable?

Input, required. An indicator of whether the user can expand the inserted row. Set the parameter to <Yes> or <No>.

Insert Mode

Input, required. An indicator of whether the new row is to be inserted into the database or to be updated into the database (assuming such a row already exists in the database), if the parent child control has a business view. If the insert mode is Update and no such row exists in the database, the system will fail to process this row. If the insert mode is Insert and a row already exists in the database, the system will fail to process the row in the database. Set the parameter to <Insert> or <Update>.

Additional Notes

You can insert the row before or after the target node. Alternatively, you can insert the row as a child of the target node. In that case, you can choose to insert it as the first or the last child.[Insert Grid Buffer Row](#)

Set Action

Use this system function to enable or disable a specific context-based action in Product Synch mode.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Action ID

Input, required. The ID of the action to enable or disable. Set the parameter to <Enabled>, <Disabled>, or an applicable object from the object list.

State

Input (EVDT_INT), required. An indicator of the state (enabled or disabled) of the action. Set the parameter to <Selected> (1), <Deselected> (0), or applicable object from the object list

Additional Notes

Note that State can be enabled, disabled, or you can pass in an EVDT_INT variable which will be interpreted to provide the value for the selection state. This makes it possible to reduce if/then logic because one system function can be used to update the enabled state for a context based action.

Set Data Dictionary Item

Use this system function to override form controls and grid columns that are data dictionary (DD) items. This is a complete change; you substitute a different DD item for the existing one. You can also create a new DD item that is not the same type as the old item.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Column

Input, optional. The column to hide. Because parent child controls have only one column, you do not need to enter a value into this parameter. Set to <All Columns>.

DD Alias

Input, required. The alias of the DD item that you want to use. Set the parameter to a specific DD alias (<Pick DD Item>) or an applicable object from the object list

System Code

Input, required. The system code to use when checking for textual overrides to apply to the DD item. Set to <default>.

Set Data Dictionary Overrides

Use this system function to change a specific DD item property.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Column

Input, optional. The column to hide. Because parent child controls have only one column, you do not need to enter a value into this parameter. Set to <All Columns>.

Overrides

Input, required. The override to apply. Set the parameter to the specific type of override to apply (<Data Dictionary Overrides>).

Set Drag Cursor

Use this system function to assign an image to the cursor when the user performs a drag operation.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Cursor

Input, required. The cursor to apply. Set the parameter to <Default> or <No Drag Cursor>.

Set Grid Cell Error

Use this system function to set an error on a cell, a tree node, a row, a column or the entire parent child control. This system function does not take hidden rows into account.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Row

Input, required. The row on which to set the error. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, or <Currently Selected Row>.

Column

Input, required. The column on which to set the error. Set to <All Columns>.

Error Code

Input, required. The error to set. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list

Set Grid Color

Use this system function to set a color on a cell, a tree node, a row, a column, or the entire control. You can select a specific color to set, or you can reset the color, which returns the color of the object to its default value. This system function does not take hidden rows into account.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row on which to set the color. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, applicable object from the object list.

Column

Input, required. The column on which to set the color. Set to <All Columns>.

Color

Input, required. The color to display. Set the parameter to a color from the color palette (<Pick Color>) or the default color <Reset Color>.

Set Grid Column Heading

Use this system function to change the text in a column heading.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Column

Input, required. The column for which to change the header. Set to <All Columns>.

Text

Input, required. The text to use for the column header. Set the parameter to alphanumeric constant (<Literal>), <Blank>, <Zero>, or applicable object from the object list.

Set Grid Font

Use this system function to set a font for the text in a cell, a tree node, a row, a column, or the entire control. Font in this context includes font type (such as Arial or Times New Roman), style (such as italic or bold), size, effects (strikeout or underline), and color. You can select a specific font to set, or you can reset the font which returns the font of the object to its default value. This system function does not take hidden rows into account.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row for which to set the font. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, applicable object from the object list.

Column

Input, required. The column on which to set the font. Set to <All Columns>.

Font

Input, required. The font to be set on the row and column. Set the parameter to a font and related settings from the Font dialog (<[Pick Font](#)>) or the default font <[Reset Color](#)>. [Insert Grid Buffer Row](#)

Set Grid Row Bitmap

Use this system function to manipulate the bitmap icon that appears next to the row. This function sets a specific system bitmap, such as a check mark or a trash can, to use as an icon on a specified row header. This system function does not take hidden rows into account.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row for which to set the bitmap. Set the parameter to an alphanumeric constant (<Literal>), <All Rows>, <Currently Selected Row>, applicable object from the object list.

Bitmap

Input, required. The bitmap to be set on the row. Set the parameter to the specific bitmap that you want to apply (<Checkbox>, <X Mark>, and so forth).

Set QBE Column Compare Style

When the application performs a QBE search, this system function permits the application to set which comparison operator to use. For example, if you want to set up a date column QBE field to be > January 1, 2005, you would use QC WorkDate = "010105" and Set QBE Column Compare Style(FC Parent/Child, GC WorkDate, <Greater Than>).

Parameters

Parent Child

Input, required. The parent child FC to affect.

Column

Input, required. The column on which to set the QBE comparison operator. Set to <All Columns>.

Compare Style

Input, required. The comparison operator to use. Set the parameter to <Equal To>, <Not Equal To>, <Greater Than>, <Less Than>, <Greater Than or Equal To>, or <Less Than or Equal To>.

Set Tree Bitmap Scheme

Use this system function to assign bitmap icons to represent the different states that a tree node might be in (open/expanded, closed/collapsed, or leaf/no children).

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Open Bitmap

Input, required. The icon to use when the node is in an expanded state; that is, all of its children are displayed. Set the parameter to <Choose Tree Bitmap> or <Default>.

Closed Bitmap

Input, required. The icon to use when the node is in a collapsed state; that is, all of its children are hidden. Set the parameter to <Choose Tree Bitmap> or <Default>.

Leaf Bitmap

Input, required. The icon to use when the node has no children and therefore cannot be expanded or collapsed. Set the parameter to <Choose Tree Bitmap> or <Default>.

Set Tree Node Bitmap

This system function replaces the default bitmap for a node with one of your choosing.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, required. The node or row for which to replace the bitmap. This value cannot be 0. Set the parameter to <Currently Selected Node>, <Last Inserted Node>, <Currently Expanding/Collapsing Node>.

Bitmap

Input, required. The bitmap image to display. Set the parameter to <Choose Tree Bitmap> or <None>.

Overlay?

Input, required. A flag that indicates whether to superimpose the image on the base bitmap image. Overlays are displayed on Windows clients only. Set the parameter to <Yes> or <No>. [Set Tree Node Clickable Bitmap](#)

Set Tree Node Bold

This system function sets the text associated with a tree node in bold type face.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, required. The tree node to affect. Set the parameter to <Currently Selected Node>, <Last Inserted Node>, <Currently Expanding/Collapsing Node>.

Bold

Input, required. A flag that indicates whether to apply or remove bold type face formatting to the text of the indicated node. Set the parameter to <Yes> or <No>.

Set Tree Node Clickable Bitmap

This system function displays an additional node bitmap to the specified tree node or row. The bitmap is clickable (clicking the bitmap causes the event to fire: **Tree Node Bitmap Is Clicked**). The additional bitmap appears to the left of the regular bitmap, so after this system function is called, the tree node has two bitmaps.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, required. The node or row to which to add the clickable bitmap. This value cannot be 0. Set the parameter to **<Current Selected Node>**, **<Last Inserted Node>**, or an applicable object from the object list

Bitmap

Input, required. The bitmap image to display. Set the parameter to **<Choose Tree Bitmap>** or **<None>**.[Set Tree Node Bitmap](#)

Set Tree Node Handle

Use this system function to set the handle of a target node.

Parameters

Parent/Child

Input, required. The parent child FC to affect.

Node

Input, optional. The ID of the node for which to get the handle. Leave this parameter blank if you want to identify the node by its position (Index parameter). Set the parameter to <Current Selected Node>, <Last Inserted Node>, <Root Node>, or <Currently Expanding/Collapsing Node>.

Index

Input, optional. The position of the node for which to get the handle. Leave this parameter blank if you want to identify the node by its condition, such as currently selected, last insert, and so forth (Node parameter). Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list

Handle

Input, required. The value to use for the handle for the target node. Set the parameter to an applicable object from the object list.

Set Tree Root Node ID

This system function assigns a node ID value to the hidden root node. You can use this system function only if the control has a node ID column.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Node ID

Input, required. A variable/control with compatible data type (same type as the node ID column) that holds a value. A type mismatch will be checked by the runtime engine. Set the parameter to an applicable object from the object list.

Show Grid Column

Use this function to show the entire column.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Column

Input, required. The column to show. Set to <All Columns>. [Hide Grid Column](#)

Show N Levels

Use this system function to expand or collapse a tree branch or the entire tree to a uniform hierarchical level.

Parameters

Parent Child

Input, required. The parent child FC to affect.

Node/Row

Input, required. The ID of the node or row under which you want to affect. Select **<Root Node>** if you want to affect the entire tree. Set the parameter to **<Root Node>** or an applicable object from the object list.

Level

Input, required. The hierarchical level to which to expand or collapse the target node or row's children. Set the parameter to an applicable object from the object list.

Suppress Fetch On Node Expand

Whenever a node is expanded, the system function, **Suppress Fetch on Node Expand**, is called from the event, **Tree Node Is Expanded**. This function tells the runtime engine not to do any fetches because ER will handle the loading of the child nodes.

Parameter

Parent Child

Input, required. The parent child FC to affect.

Suppress Grid Line

Use this function to prevent a row from becoming part of the grid. For example, use this system function on the **Write Grid Line Before** event to prevent the line from being written to the grid.

Parameter

Parent/Child - Grid

Input, required. The parent child FC to affect.

Suppress Node Indent/Outdent

This system function prevents users from being able to move nodes horizontally within the tree structure.

Parameter

Parent Child

Input, required. The parent child FC to affect.[Suppress Node Move Up/Down](#)

Suppress Node Move Up/Down

This system function prevents users from being able to move nodes vertically within the tree structure.

Parameter

Parent Child

Input, required. The parent child FC to affect.[Suppress Node Indent/Outdent](#)

Update Grid Buffer Row

Use this system function to update a row from the GB into the grid control. This system function does not take hidden rows into account.

Parameters

Parent/Child - Grid

Input, required. The parent child FC to affect.

Row

Input, required. The relative row to affect. Set the parameter to an alphanumeric constant (<Literal>), <Currently Selected Row>, <Currently Expanding/Collapsing Node>, or an applicable object from the object list.

Selectable?

Input, required. An indicator of whether the user can select the updated row. Set the parameter to <Yes> or <No>.

Protected?

Input, required. An indicator of whether the user can edit the updated row. Set the parameter to <Yes> or <No>.

Updateable?

Input, required. An indicator of whether runtime will attempt to update the underlying table if the user edits the data and clicks the OK button. Set the parameter to <Yes> or <No>.

Deleteable?

Input, required. An indicator of whether the user can delete the updated row. Set the parameter to <Yes> or <No>.

Clear After?

Input, required. An indicator of whether runtime should clear the GB automatically immediately after the update. Set the parameter to <Yes> or <No>.

Was Grid Cell Value Entered

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

Parameters

Parent Child

Input, required. The parent child FC to affect.

Column

Input, required. The column containing the grid cell to check. Set to <All Columns>.

Return To

Input, required. The object to which to assign the return value that designates whether the cell or node has changed. Set the parameter to an applicable object from the object list.

Was Grid Cell Value Entered

26

Understanding Push Button Controls

This chapter contains the following topics:

- [Section 26.1, "Push Button Controls"](#)
- [Section 26.2, "Push Button Events"](#)

26.1 Push Button Controls

Use a push button to initiate an action or a set of actions. You can designate a single push button to be the one that is activated when the user presses Enter by enabling the Default Pushbutton property.

Message forms have a push button on them by default. The button is configurable.

Subforms and message forms do not have a tool bar; instead, users must use push buttons. The push buttons behave the same as the standard push buttons for other form types.

26.2 Push Button Events

These events are the only ones that can fire when the user clicks a push button:

- Button Clicked
- Post Button Clicked
- Post Button Clicked - Asynch

The first two always fire when a user clicks a button, and they fire in succession. That is, **Post Button Clicked** fires after runtime executes any logic that might exist on **Button Clicked**. **Post Button Clicked - Asynch** fires only when the button is an OK button.

Understanding Radio Button Controls

This chapter contains the following topics:

- [Section 27.1, "Radio Button Controls"](#)
- [Section 27.2, "Radio Button Design-Time Considerations"](#)
- [Section 27.3, "Radio Button Events"](#)

27.1 Radio Button Controls

Use radio buttons to indicate choices. Selecting a radio button indicates which function in a set of functions is to be enabled. Radio button option sets should always be mutually exclusive. Enclose a set of radio button controls in a group box control to make them function as a single unit.

You can associate radio buttons with a user-defined code (UDC) field, where each button has a value from the UDC table. You must associate a radio button with a database or data dictionary (DD) item.

27.2 Radio Button Design-Time Considerations

A standard use case for a group of radio buttons is to provide users with a choice of operators to use in a query. If you want to use radio buttons for this purpose, use the Filter Criteria property options.

By default, the control returns a Boolean numeric value to indicate its state (0 for deselected and 1 for selected). However, you can change the return value for a radio button control with the Value property.

27.3 Radio Button Events

Only one event occurs on a radio button control: **Selection Changed**. Runtime fires the event when the user changes the status of a radio button. Consequently, **Selection Changed** should occur in pairs for radio buttons: Once for the selected control and once for the control that the system deselects.

28

Understanding Static Text Controls

This chapter contains the following topics:

- [Section 28.1, "Static Text Controls"](#)

28.1 Static Text Controls

Use a static text control to display descriptive text, such as a title or instructions. This text is not associated with a control, and the user cannot change it. You can change it during runtime using the **Set Control Text** system function in event rules. You also can make static text clickable by enabling the Clickable property. When a user clicks clickable text, runtime fires the **Text Clicked** event.

No runtime processing occurs on this control.

29

Using Subforms and Subform Aliases

This chapter contains the following topics:

- [Section 29.1, "Understanding Subforms"](#)
- [Section 29.2, "Understanding Subform Design-Time Considerations"](#)
- [Section 29.3, "Understanding Subform Events"](#)
- [Section 29.4, "Understanding Subform Runtime Processing"](#)
- [Section 29.5, "Creating Subforms"](#)
- [Section 29.6, "Reusing Subforms"](#)
- [Section 29.7, "Working with Data Structures and Subforms"](#)
- [Section 29.8, "Working with Functions and Subforms"](#)
- [Section 29.9, "Subform System Functions"](#)

29.1 Understanding Subforms

A subform is a control designed for use on a power form or another subform. Although technically a control, subforms have some form characteristics as well. Each subform represents one data view; that is, each subform control can have a single business view (BV) attached to it. Power forms can contain several subforms, so a single power form with multiple subforms enables users to see multiple data views. For example, a user selecting a purchase order in a grid could see related shipping information in one subform and fulfillment information in another subform on the same power form. When the user selects a row in the grid, all of the data is updated and, most importantly, the user does not have to open a new form to see the updated data.

Together, power forms and subforms provide developers with the ability to code:

- Multiple views on a form.
- Multiple grids on a form.
- Multiple tab controls on a form.
- Tab pages with their own business view.
- BVs that can communicate with each other, and even react to selection and data changes that occur in other views on the form.

Subforms have two main characteristics:

- To its parent, it is a control.

- To the controls that it contains, it is a form.

When you place subforms on a power form or subform, the system treats the interactions among them as parent/child relationships, with the power form or subform as the parent and the subforms as the children of the power form and siblings to each other. This hierarchical relationship governs logical relationships. It is also represented visually in the Form Design Aid (FDA) Application Tree View as a hierarchy to help you understand the relationships.

Similarly to a form, a subform owns the data flows between the interface view and the database, including browse, insert, update, or delete. In fact, you create a subform as if it were a form, and then place it as a control. Do not let its appearance fool you, however. A subform is really a control and the FDA interface treats it accordingly. For example, if subform B is contained in power form A, when you select B, the FDA control bar still displays the title of A. Additionally, if you view the data structure, you will see the data structure for A and not for B.

29.2 Understanding Subform Design-Time Considerations

This list describes many of the conditions and factors that you should take into account while designing the subform in FDA:

- Subforms have their own BVs (only one per subform), grids, and filters that pass logic between other subforms.
- Subforms can be used as tab pages, or they can be placed on a tab page.
- Multiple tab controls are permitted unless the subform is inside a tab page.
If the subform is inside a tab page, it cannot have tab controls. However, this restriction cannot be enforced programmatically when you reuse subforms. Therefore, when developing reusable subforms, carefully consider the context in which you expect to use them to prevent poor user interface design.
- Toolbar and form/row exits are not permitted on a subform.
- Subforms do not contain scroll bars; you must display all controls within the form.
- You can add action buttons (Cancel, OK, and so forth) to a subform.

No default action buttons are defined by FDA. Default actions can be placed anywhere on the subform. For example, the default action, Save, can be placed beneath the grid. Use the dotted lines at the top and bottom of a subform during design time as guides for where to place more common default actions. For example, buttons such as Select or Find belong at the top of the form while buttons such as Save or Cancel belong at the bottom.

- You can define ER that fires when the user sets or removes focus from the subform, using the **Enter Focus** and **Leave Focus** subform events.

After designing the subform, you must return to the parent form and map the parent data fields to the child so that the two can share data as you intend. After associating a data structure with the parent, use the Mapping Links property to establish the data mapping between the parent and subform.

Note: Because subforms can act as parents, they have the Mapping Links property. However, you can only create mappings on the parent level. If you cannot find the child subform in the Link To field, then you might have mistakenly entered the data mapping property for the child instead of the parent.

These property values are significant for subforms:

- Show Subform Header

This property enables you to hide or show the title bar of the subform. You must show the header if you want to make the subform collapsible (with the Collapsible property).

- Collapsible

This property determines whether the user can choose to view just the title bar (header) of the subform. The ability to collapse subforms is useful on forms that contain a large number of subforms. You must show the header (with the Show Header property) if you want to make the subform collapsible.

- Transaction

This property determines where the subform falls within the transaction boundary for the form as a whole.

Additionally, subforms support the form-level properties: **Update on Businessview** and **Fetch on Businessview**.

Finally, to enhance reuse, you can associate any number of functions with a subform at design time. Then, during runtime, the parent of the subform can invoke any of those functions with the **Call Function** system function.

See Also: ■[Working with Data Structures and Subforms](#).

- [Working with Functions and Subforms](#).

29.3 Understanding Subform Events

These events can fire on the subform during runtime:

- Notified by Child
- Notified by Parent
- Enter Focus
- Leave Focus
- Tab Page is Initialized
- Tab Page is Selected
- Row Is Selected
- Grid Record is Fetched
- Write Grid Line-Before
- Write Grid Line-After
- Row Is Unselected
- Last Grid Record Has Been Read
- Add Record to DB - Before
- Add Record to DB - After
- Update Record to DB - Before
- Update Record to DB - After
- Post Commit

Notified by Child and **Notified by Parent** fire when the child sends data to the parent and vice versa, respectively. Use the **Notified by Parent** event to run any business logic that you want the application to execute after the parent of the subform calls the **Notify Child** system function. This logic is usually based on the information passed to this subform from its parent. Use the **Notified by Child** event to run any business logic that you want the application to execute after its child calls the **Notify Parent** system function. This logic is usually based on the information returned from its child. **Enter Focus** enables you to run ER when the user sets the focus to a subform by entering that subform with a tab or a mouse click. You can use **Leave Focus** to run ER when the user sets the focus outside of a subform by leaving that subform with a tab or a mouse click.

In addition, a number of events that start with **WIZARD** can fire on this control as well, but only when on a wizard form.

See Also: ■[Understanding Wizard Controls](#).

29.4 Understanding Subform Runtime Processing

You use subforms to browse or update records using one BV; you cannot place more than one BV on a subform. Subforms can contain most FDA controls (with the notable exception of the parent child control) and may or may not have a grid.

This section discusses how runtime processes the subform control.

29.4.1 Control Initialization

Each subform ultimately resides on a parent power form. When the power form initializes, all of its subforms (including the child subforms of its subforms) are initialized. First, runtime initializes these objects in order:

- Business view columns (BC).
- Subform controls.
- Static text.
- Helps.
- Event rules (ER) structures.

Runtime begins detail data selection and sequencing if the grid option **Automatically Find On Entry** is enabled.

29.4.2 Subform Push Buttons

Subforms do not have a tool bar; instead, you must add push buttons to perform "standard" functions. The push buttons behave the same as the standard push buttons for other form types.

If a subform is on a power browse form, these push buttons are available:

- Select
- Find

If a subform is on a power edit form, these push buttons are available:

- Clear
- Find
- Delete

- Save

29.4.2.1 Find

Find is a standard push button that is available on all subforms. When the user clicks it, runtime fires the **Button Clicked** event. If no errors exist in the filter fields, runtime performs data selection and sequencing for the grid or other control if no grid is present. If one or more records are fetched, then runtime loads the control, and sets the mode to Update. If no records were fetched and a grid is present, runtime sets the mode to Add. Then runtime fires the **Post Button Clicked** event.

29.4.2.2 Select

Select is a standard push button that is available on subforms placed on power browse forms. When clicked, runtime performs these actions:

1. Load the mapping link value from the subform parent.
2. Fire **Button Clicked**.
3. Fire **Post Button Clicked**.

29.4.2.3 Clear

Clear is a standard push button that is available on subforms placed on power edit forms. When clicked, runtime performs these actions:

1. Fire **Button Clicked**.
2. Clear all controls on the subform.
3. Clear errors on the subform.
4. Fire **Post Button Clicked**.

29.4.2.4 Delete

Delete is a standard push button that is available on all subforms placed on power edit forms. The actual delete from the database does not occur at this point. Runtime verifies the intention to delete when the user clicks the Delete button, and then commits the deletion when the user clicks the Save button. Consequently, if the user clicks the Cancel button, the records are not purged from the database.

29.4.2.5 Save

Save is a standard push button that is available on all subforms placed on power edit forms. It validates the information on the subform and updates or adds to the database.

Note: If the subform contains a grid control, the save button processing behaves similarly to the OK button on a headerless/detail form. If the subform does not contain a grid, the Save button processing behaves similarly to the OK button on a fix/inspect form.

See Also: ■[Fix/Inspect Runtime Processing](#).

- [Header Detail Runtime Processing](#).

29.5 Creating Subforms

This section provides an overview of subform creation, and discusses how to:

- Create a subform without a power form.
- Create a subform on a power form.
- Create a subform as a tab page.

29.5.1 Understanding Subform Creation

You can create a subform in one of two ways: without a power form, or as a control either directly on a power form or on the tab page of a power form. These two options give you the flexibility to code subforms and power forms independently to accommodate different design schedules. Therefore, you might choose to create a subform outside of the context of a power form if the power form itself is not ready yet or if your group is responsible only for creating a subform that other groups will use on their power forms, for example. Furthermore, you can only create browse subforms directly on a power form, so if you want to create an edit subform, you must create it independently of a given power form.

No matter how you create the subform, it exists as its own entity in the system. Therefore, you can find and insert any subform onto any power form.

When you create it, a blue hashed line appears at the top and bottom of a subform. Use these lines as guides for button placement. Place initial actions, such as find, at the top.

Place concluding actions, such as save, near the bottom. Controls may only exist within the boundary of the subform; you cannot size the subform to be smaller than the area where its child controls reside. This restriction includes hidden controls, so if FDA does not permit you to shrink a subform, ensure that you have no hidden controls preventing the resize.

29.5.2 Creating a Subform without a Power Form

To create a subform without a power form:

1. In FDA, select a subform type (reusable browse or reusable edit) from the Form menu under Create.
2. On Subform Properties, configure the properties for the subform.

Note: When you save the subform, you are actually creating an application to contain the subform. Keep that in mind when you later insert an alias to the subform on a power form. When you search for the subform, you must first search for the application that contains the subform, and then you can select the subform itself.

29.5.3 Creating a Subform on a Power Form

To create a subform on a power form:

1. In FDA, open the power form that you want to use to contain the subform.
2. If you want to insert the subform on a tab page, click the tab control to insert into.
3. Select Subform from the Insert menu.
4. On Subform Properties, configure the properties for the subform.

5. If you are inserting the subform directly on the power form (and not on a tab page), click to place the subform where you want it on the power form (as you would any other control).

If you later change your mind, you can always drag the subform from a tab page to the main power form or vice versa.

29.5.4 Creating a Subform as a Tab Page

To create a subform as a tab page:

1. In FDA, open the form to which you want to add tab pages.
2. Add a tab page to the form.
3. Select the tab page and select Subform or Subform Alias from the Insert menu.

Alternatively, drag a subform from another position on the parent form and drop it on the tab page.

29.6 Reusing Subforms

This chapter provides an overview of subform reuse and discusses how to reuse a subform on a power form.

29.6.1 Understanding Subform Reuse

Subforms can exist as discrete objects in the system. You can place the subform itself on a power form; this action is referred to as *embedding*. You can also reference an existing subform on a power form with an *alias*; this action is referred to as *reusing*. Typically, developers reuse subforms that are designed to appear on numerous forms, such as the display of address book information. In this way, you can more easily standardize the interface.

A reused subform is actually just a pointer; therefore, it is unaware of its parent and children in any given context. If you want a reused subform to communicate with its parent or children, you must do so through ER. While you cannot embed a subform within another subform, you can reuse a subform within another subform. In other words, you can embed or reuse a subform within a reusable subform, but you cannot embed or reuse a subform within an embedded subform.

Inserting reusable subforms in a form is different from inserting an embedded subform. They exist as two different control types in the user interface. Therefore, the FDA Menu/Toolbars contain two different insert actions. If you insert an alias, you are prompted for the application. If you insert a reusable subform, you are prompted for the subform. You cannot insert a subform onto a form until the application the subform has been defined in has been saved.

To make reusing subforms effective, you must plan carefully and be cautious when altering subforms. For example, if both a parent and a child are reused, then you must set up their data mappings in a way that facilitates their passing information to each other. You should use variables as often as possible, especially for business view columns (BCs), grid columns (GCs), and form controls (FCs). Establish a naming convention for the variables so that you can determine which ones were created for the purpose of mapping. That way, if another developer wants to reuse the subform and sees several "extra" variables, that developer will understand that the variables are not extraneous.

Knowing that any subform can be reused, you must be careful when changing a subform, even just resizing it. Anyone who is currently pointing to the subform will see the changes you make. It is possible that if you make the subform a little larger that it will no longer fit on someone else's form. Even worse, if you change the data structure by removing elements from it, you might break someone else's application.

Note: Always determine the full effect of changing a subform, especially its data structure. If you reuse subforms in the applications, check them occasionally to ensure that the subforms have not been altered in such a way as to negatively affect the applications.

29.6.2 Reusing a Subform on a Power Form

To insert (reuse) a subform on a power form:

1. In FDA, open the power form that you want to use to contain the subform.
2. If you want to insert the subform on a tab page, click the tab control to insert into.
3. Select Subform Alias from the Insert menu.
4. On Work with Applications, select the application containing the subform you want to insert.
5. On Work with Subforms, select the subform you want to insert.

The system inserts an alias to the subform on the power form or tab page.

Note: Always determine the full effect of changing a subform. If you reuse subforms in the applications, check them occasionally to ensure that the subforms have not been altered in such a way as to negatively affect the applications.

29.7 Working with Data Structures and Subforms

Subforms are self-contained; the elements within it cannot be accessed by other subforms or the form. Therefore, to communicate with each other, each subform must present an interface that can be used to pass data into and out of the subform. The interface is presented as a data structure and performs similarly to form and report interconnects. When developing event rules, the data structure items will appear as variable type subform interconnection (SI).

29.7.1 Mapping a Parent's Variables to a Child Subform

To map parent variables to a child subform:

1. Select the parent and select Form Properties from the Form menu.
2. Click the Mapping Link tab.
3. In the Link To field, select the child subform to which you want to map data.
You cannot map to a grandchild.
4. Select an element of the child interface and map a variable to it.

Note: Changing the parent assignment after mapping links have been set or event rules have been developed could cause the application to fail.

29.8 Working with Functions and Subforms

To enhance reuse, you can associate any number of functions with a subform at design time. Then, during runtime, the parent of the subform can invoke any of those functions with the **Call Function** system function.

29.8.1 Adding a Function to a Subform

To add a function to a subform:

1. Click the subform to which you want to add the function and select Form, Create, Function .

The Application Tree View browser switches to the Logical Hierarchy view, and the function appears in the tree as a child object of the subform. (Functions do not appear in the tree if set to the Physical Hierarchy view.)

2. Right-click the function in the tree and select Edit ER.

The Event Rules Design form appears.

3. Create the logic for the function as you would for an ER.

29.9 Subform System Functions

This section describes the system function unique to subforms.

Call Function

This system function enables the parent to invoke any of the functions on any of its children.

Parameters

Subform

Input, required. The subform to affect.

Function

Input, required. The function to invoke. Set the parameter to an applicable object from the object list.[Working with Functions and Subforms](#)

Enable Subform

This system function makes the subform available for user entry and system use.

Parameter

Subform

Input, required. The subform FC to affect. If the subform has children, you can enable all of its children as well.

Disable Subform

This system function makes the subform unavailable for user entry and system use. You cannot invoke functions on a disabled subform.

Parameter

Subform

Input, required. The subform FC to affect. If the subform has children, you can disable all of its children as well.

Hide Subform

This system function hides the subform from the user, although the system can still access it. For example, you can invoke functions on a hidden subform.

Parameter

Subform

Input, required. The subform FC to affect. If the subform has children, you can hide all of its children as well.

Show Subform

This system function displays a previously-hidden form to the user.

Parameter

Subform

Input, required. The subform FC to affect. If the subform has children, you can show all of its children as well.

Update Parent

This system function causes the data in the data structure to be distributed appropriately. It has no parameters.

Notify Parent

This system function triggers the **Notified by Child** event. It has no parameters.

Get Error Count

This system function returns the number of errors that are set on a subform, including its children, if desired.

Parameters

Subform

Input, required. The subform FC to affect. If the subform has children, you can include the errors set on them in the count as well.

Number

Input, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the total number of errors set on the objects indicated. Runtime returns the value to the object indicated by Number.

Get Warning Count

This system function returns the number of warnings that are set on a subform, including its children, if desired.

Parameters

Subform

Input, required. The subform FC to affect. If the subform has children, you can include the warnings set on them in the count as well.

Number

Input, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the total number of warnings set on the objects indicated. Runtime returns the value to the object indicated by Number.

Get Subform ID

This system function acquires the ID of the child subform relative to the current form.

Parameter

Subform ID

Input, required. The object to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the ID of the child subform to the object specified by Subform ID.

Notify Child

This system function triggers the **Notified by Parent** event.

Parameter

Subform

Input, required. The child that you want to contact. Set the parameter to <All Children> or an applicable object from the object list.

Trigger Default Action

This system function enables you to "push a button" on a subform programmatically.

Parameters

Subform

Input, required. The subform FC to affect.

Default Action

Input required. The "button" to "push." Set the parameter to <Subform Save>, <Subform Delete>, <Subform Find>, <Subform Clear>, <Subform Select>.

Expand Subform

This system function expands a subform if it is currently collapsed. It has no effect if the Collapsible property is disabled.

Parameter

Subform

Input, required. The subform FC to affect. Set the parameter to <Current Subform>, <All Children>, or an applicable object from the object list.

Collapse Subform

This system function collapses a subform if it is currently expanded. It has no effect if the Collapsible property is cleared.

Parameter

Subform

Input, required. The subform FC to affect. Set the parameter to **<Current Subform>**, **<All Children>**, or an applicable object from the object list.

Collapse Subform

30

Understanding Tab and Tab Page Controls

This chapter contains the following topics:

- [Section 30.1, "Understanding Tab and Tab Page Controls"](#)
- [Section 30.2, "Creating Tab Controls"](#)
- [Section 30.3, "Tab Control System Functions"](#)

This chapter provides an overview of tab and tab page controls, and discusses how to create tab controls.

30.1 Understanding Tab and Tab Page Controls

You can create a control that enables you to split a form into different tabbed pages. Tabs enable you to use multiple controls on a single form. You can group the control functions by placing related controls on different tab pages for a single form. You can cut and paste controls from one page to other pages.

The form has a single business view (BV). One commit for the form exists on the OK button. You can use system functions such as **Set Focus** to add additional functions for the tab controls. Each tab page has a **Tab Page is Selected** event and a **Tab Page is Initialized** event associated with it. You can attach additional event rule logic to these events. When you use tab pages in an application, you can focus on the upper-right corner of the tab page and move it around. This strategy enables you to see several pages at the same time.

Additionally, you can embed or reuse subforms on a tab page, or you can specify a subform to act as the tab page itself.

30.2 Creating Tab Controls

This section discusses how to create a tab control.

30.2.1 Creating a Tab Control

To create a tab control:

1. On the form with which you are working, select Tab Control from the Insert Controls tool bar.

Page Properties appears. It indicates which page of information you are on.

2. Complete the Event Rules Title.

The form appears with a tab at the top, named as you indicated.

3. Position and resize the control.
4. Select Tab Page from the Insert Controls tool bar to add additional tab pages, as necessary.

The size of each page in the tab control is equal to the size of the entire tab control. You cannot resize an individual page to be bigger or smaller than the others.

All tab pages appear as children of the tab control in the Application Tree View.

5. Add controls to the tab pages as if they were individual forms.

30.3 Tab Control System Functions

These system functions are specifically applicable to tab controls. They are located in the Control folder.

Disable Tab Page

Use this control to render all controls on a tab page unavailable for entry both by the end user and programmatically. A disabled control is still visible.

Parameters

Tab Control

Input, required. The tab form control (FC) to affect.

Tab Page

Input, required. The tab page to disable. Set the parameter to a tab page from the list of objects.[Enable Tab Page](#)

Enable Tab Page

Use this system function to render all controls on a tab page available for entry both by the end user and programmatically.

Parameters

Tab Control

Input, required. The tab FC to affect.

Tab Page

Input, required. The tab page to enable. Set the parameter to a tab page from the list of objects.[Disable Tab Page](#)

Hide Tab Page

Use this system function to prevent the user from seeing (and therefore interacting with) the controls on a tab page. Hidden controls can be manipulated programmatically.

Parameters

Tab Control

Input, required. The tab FC to affect.

Tab Page

Input, required. The tab page to hide. Set the parameter to a tab page from the list of objects.

Additional Notes

This function can only be called on **Post Dialog is Initialized**. At all other times, use **Disable Tab Page** instead.[Disable Tab Page](#)

Set Current Tab Page

Use this system function to programmatically "click" a tab page, thereby bringing it to the forefront and making it active.

Parameters

Tab Control

Input, required. The tab FC to affect.

Tab Page

Input, required. The tab page to display. Set the parameter to a tab page from the list of objects.

Set Tab Page Text

Use this system function to change the title of a tab page in a given instance.

Parameters

Tab Control

Input, required. The tab FC to affect.

Tab Page

Input, required. The tab page for which to change the title. Set the parameter to a tab page from the list of objects.

Text

Input, required. The text to show as the label for the control. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

31

Understanding Text Block Controls

This chapter contains the following topics:

- [Section 31.1, "Text Block Controls"](#)
- [Section 31.2, "Charts in Text Blocks"](#)
- [Section 31.3, "Text Block Control Design-Time Considerations"](#)
- [Section 31.4, "Text Block Control Charts Design-Time Considerations"](#)
- [Section 31.5, "Text Block Events"](#)
- [Section 31.6, "Text Block Control System Functions"](#)

31.1 Text Block Controls

You can use a text block control to create different text segments and then attach attributes to them. You can format the text segments differently so that each segment looks different. For example, you can create a clickable text segment and add event rules to the **Text Clicked** event so that you can click the text to connect to a different form. You can also use several system functions with this control. The text block control is particularly useful for Web applications.

31.2 Charts in Text Blocks

You also use text blocks to create graphs like those in the Plant Manager Dashboard application. The graphs you can create are:

- Bar
- Combo
- Line
- Pie
- Pie_ontime
- Stacked_bar
- Stacked_bar_ontime

The format, fonts and color choices for each chart type are defined by their templates. You cannot change these values.

31.3 Text Block Control Design-Time Considerations

Unlike most Form Design Aid (FDA) controls, text block properties in the Property Browser do not mirror those in the properties dialog that appears when you double-click the control. Set standard properties (such as height, title, and so forth) in the browser. Create the text strings themselves with the properties dialog. For each text string in the control, you can designate whether it is clickable, and you can control its font and color. For even more control over the appearance of the content of the control, you can include HTML tags as part of the text segments. The tags do not appear at runtime, but the engine formats the control as indicated.

31.4 Text Block Control Charts Design-Time Considerations

XML template files for each chart type are included. Those templates are:

- bar_basic.xml
- combo_basic.xml
- line_basic.xml
- pie_basic.xml
- pie_ontime.xml
- stacked_bar_basic.xml
- stacked_bar_ontime.xml

You define the template you want to use in an XML file. You then insert the XML file into a text block control, and runtime uses that file to create the graph. This is an example XML file:

```
<?xml version="1.0" encoding="utf-16"?> <Graph graphName="bar_basic"> =>
<O1Title text="Week Ending" visible="true"/> =>
<Y1Title text="Production Cost Variance (USD)" visible="true"/>
<LocalRelationalData>
  <Row columnKey="1-Sept 05" rowKey="Actual Variance" dataValue="1504" />
  <Row columnKey="8-Sept 05" rowKey="Actual Variance" dataValue="980" />
  <Row columnKey="15-Sept 05" rowKey="Actual Variance" dataValue="-675" />
  <Row columnKey="22-Sept 05" rowKey="Actual Variance" dataValue="784" />
  <Row columnKey="20-Sept 05" rowKey="Actual Variance" dataValue="0" />
</LocalRelationalData>
</Graph>
```

The graphName attribute defines the type of chart you are displaying in your application.

To create the XML file, can write a business function that performs any calculations and then writes the XML to an array variable. You can also use write hard-coded strings within ER. In both of these cases, the encoding of the resulting XML file is UTF-16.

If, however, you write C code business functions and use a tool outside of the JD Edwards EnterpriseOne system to create the XML file, it is likely that the encoding will not be UTF-16. This creates problems when the chart is displayed in the browser. It is recommended that you do not create XML files using any tool or operating that does not use UTF-16 encoding. If you do, you must be sure to set the encoding in the XML file to be the same as your application.

31.5 Text Block Events

Only one event can fire on the text block control during runtime: **Text Clicked**.

31.6 Text Block Control System Functions

System functions for text block controls are located in the Text Control Functions folder.

This section discusses the system functions for text block controls.

Add Segment

Use this system function to add a text segment (clickable or not) to the text block.

Parameters

Text Control

Input, required. The text control form control (FC) to affect.

Text

Input, required. The content of the text segment to add. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Font

Input, required. The font (including type, style, color, and so forth) to apply to the text segment. Set the parameter to select a font and its properties (<Pick Font>), or to revert to the default font and property settings (<Reset Font>).

Clickable

Input, required. An indicator of whether the text segment is clickable. Set the parameter to <Yes> or <No>.

SegmentID

Output, required. The variable to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the ID of the segment you added to the object indicated by SegmentID.

Get Last Clicked Segment

Use this system function to acquire the ID of the segment that was clicked which caused the current event (**Text Clicked**) to fire.

Parameters

Text Control

Input, required. The text control FC to affect.

SegmentID

Output, required. The variable to which to assign the return value. Set the parameter to an applicable object from the object list.

Returns

This system function returns the ID of the segment just clicked to the object indicated by SegmentID.

Get Segment Information

Use this system function to acquire the textual content of a given segment, along with whether the segment is clickable.

Parameters

Text Control

Input, required. The text control FC to affect.

Text

Output, required. The variable to which to return the textual content of the text segment. Set the parameter to an applicable object from the object list.

Clickable

Output, required. The variable to which to assign the return value. Set the parameter to an applicable object from the object list.

SegmentID

Input, required. The ID of the segment to affect. Set the parameter to an applicable object from the object list.

Returns

This system function returns the text of the segment and an indicator of whether it is clickable to the objects indicated by Text and Clickable, respectively.

Remove Segment

Use this system function to delete a text segment from a given text block control.

Parameters

Text Control

Input, required. The text control FC to affect.

SegmentID

Input, required. The ID of the segment to affect. Set the parameter to an applicable object from the object list.

Update Segment

Use this system function to change text, font, color, and clickability of a given text segment in a text block control.

Parameters

Text Control

Input, required. The text control form control (FC) to affect.

Text

Input, required. The text that you want to display in the text segment. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Font

Input, required. The font (including type, style, color, and so forth) to apply to the text segment. Set the parameter to select a font and its properties (<Pick Font>), or to revert to the default font and property settings (<Reset Font>).

Clickable

Input, required. An indicator of whether the text segment is clickable. Set the parameter to <Yes> or <No>.

SegmentID

Input, required. The ID of the segment to affect. Set the parameter to an applicable object from the object list.

32

Understanding Secured Enterprise Search

This chapter contains the following topics:

- [Section 32.1, "Secured Enterprise Search"](#)

32.1 Secured Enterprise Search

The text search control within FDA is used for SES. When utilizing SES as the active text search engine the following fields are hidden at runtime.

- Case Sensitive
- Include Similar Words

When you build the business view for the form containing the text search control, you might consider including the data dictionary items, sesscr and txtsum. The sesscr data item displays the score for each match returned; that is, the extent to which the result matches that for which the user originally searched. Txtsum displays the summary for the returned value; that is, the context in which the match occurred. When placed in a grid, the system populates the column automatically when the user executes a find.

Note: You must enable the business view attached to the form for text searches using Object Management Workbench (OMW), or the text search function will not work properly on the form.

This control has no unique property settings in Form Design Aid (FDA).

You cannot apply logic to the text search control, so it has no unique system functions. No events fire in response to user or runtime interaction with the control. The interconnectivity level has no impact on this control.

Set Text Search Keyword System Function

To give flexibility to application developers to pass complex SES query based grammar as part of the keywords, the 'Set Text Search Keywords' system function is made available in the General category of system functions.

You can provide certain options like Match Any/Match All/Match Exact/Complex Query to the end user. Based on the option selected by the end user custom logic can be written to generate queries following SES rules and set that query to the system function provided.

Run Time Filters

If a run time filter option is selected from the filter options of a filter column a different set of operators is displayed based on the data type of the filter column.

Data Type	Operators
String/Var String	Contains Equals !=
Character	Equals !=
Number/Date	< > <=br/>>=br/>=br/>!=

Incremental Build

A full build results in the generation of the RSS feed for all records in the view. When a small set of records are modified (added/updated) as part of the business logic, it is not desirable to request a full build frequently. In such scenarios, an incremental build invocation can be embedded in the business function logic.

Note: The full and clear builds are generally invoked from the Index build Definition (P95800A) application.

Below is code representing an incremental build embedded in the business function logic.

```

LPVOID          lpVoid;
HENV            hEnv;
HUSER           hUser;
HREQUEST        hRequest;
KEY1_F0101      dsKeySentStruct = {0};
F0101          dsStruct = {0};
JDEDB_RESULT    rcode;
MATH_NUMERIC    mmSelect;
LPKEYINFO       lpKeyInfo = NULL;
BOOL            bTranStatus;
JCHAR           szBuf[1024] = {0};

short nNumKeys = 1;
int nNotUsed = 0;

if (JDB_InitEnv(&hEnv) != JDEDB_PASSED)
{
    printf("JDB_InitEnv Failed\n");
    return 0;
}

if (JDB_InitUser(hEnv,&hUser,_J("TEST"),JDEDB_COMMIT_AUTO) != JDEDB_PASSED)
{

```

```

        printf("JDB_InitUser Failed\n");
        JDB_FreeEnv(hEnv);
        return 0;
    }

    JDB_TextSearchOpenView(hUser,_J("V0101C"),_J("Business Data -
TEST"),&hRequest);

    lpKeyInfo = (LPKEYINFO) jdeAlloc(COMM_POOL,sizeof(KEYINFO)*1,Mem_ZeroInit);

    jdeNIDcpy(lpKeyInfo[0].szTable,_J("F0101"));
    jdeNIDcpy(lpKeyInfo[0].szDict,_J("AN8"));
    lpKeyInfo[0].idInstance = 0;
    ParseNumericStringEx (&mnSelect, _J("1"),Default_Separator);
    lpKeyInfo[0].lpJDEValue = &mnSelect;

    SesTextSearchIncrementIndexing(hUser,_J("V0101C"),_J("Business Data -
TEST"),TEXTSEARCH_INDEX_INSERT,lpKeyInfo,1);

    jdeFree(lpKeyInfo);

    JDB_FreeUser(hUser);

    JDB_FreeEnv(hEnv);
}

```

The SES Text Search Increment Indexing elements are described in the table below.

Value	Direction	Element
hUser	->	User Handle
_J("V0101C")	->	Business View Name
_J("Business Data - TEST")	->	Data Source Override
TEXTSEARCH_ INDEX_INSERT	->	Index Mode
lpKeyInfo	->	Key Info
1	->	Number of Keys

The options for Index Mode are:

- TEXTSEARCH_INDEX_INSERT
- TEXTSEARCH_INDEX_UPDATE

33

Understanding Tree Controls

This chapter contains the following topics:

- [Section 33.1, "Tree Controls"](#)
- [Section 33.2, "Tree Control Events"](#)
- [Section 33.3, "Tree Control System Functions"](#)

33.1 Tree Controls

Tree controls display data in a hierarchical format. You can have multiple controls on a single form if you need more than one tree.

The tree itself is assembled from data that you have placed into cache. The data structure you create for this purpose must load at least three columns into cache:

- Node Value

This value is used primarily for placing the row as a node in the tree. Every node in the tree requires a value which corresponds to its position in the tree, based on parent node value. The secondary purpose of node value is for identification.

- Node Description

This value is used to label each node in the tree for the benefit of the users.

- Parent Value

This value is used to indicate which row is its parent. The value corresponds to Node Value. Runtime adds this row as a child node of its parent in the tree.

Based on this data, and given which node value is to be used as the parent, runtime can construct a tree. For example, consider this table of data:

Node Value	Node Description	Parent Value
1	Alpha Manufacturing	0
2	Branch A	1
3	Branch B	1
4	Plant x	2
5	Plant y	2
6	Plant z	3

Given that the root node value is 1, runtime could assemble a tree with this structure:

```
Alpha Manufacturing
  Branch A
    Plant x
    Plant y
  Branch B
    Plant z
```

Consequently, you must ensure that the data in the node value and parent value columns correspond such that runtime can derive the hierarchy. Additionally, many of the system functions that you can use to manipulate the tree rely on node value as an identifier. Therefore, you might want to ensure that those values are unique.

You can also use table I/O to provide node data by associating a given node with a particular table handle using the **Set Tree Node Handle** system function.

Tree controls have no control-specific properties in Form Design Aid (FDA). Additionally, no automatic runtime processing occurs for the control. Tree controls are manipulated by system functions exclusively.

33.2 Tree Control Events

These events can fire on the tree control during runtime:

- Set Focus On Tree
Fires when the tree control acquires focus.
- Tree Node Selected
Fires when the user clicks a tree node.
- Tree Node Is Expanding
Fires when the user or engine expands a node. This event fires only on the first expand incident for a particular node in a session.
- Tree Node Is Collapsing
Fires when the user or engine collapses a node.
- Double Click on Leaf Node
Fires when the user double-clicks a leaf node.
- Get Custom Tree Node
Fires when the user or engine expands a node for the first time in the session. It also fires if page-at-a-time processing is enabled and the user loads a new page.
- Tree Node Is Deleted
Fires when the user or engine deletes a tree node.
- Kill Focus On Tree
Fires when the tree control loses focus.

33.3 Tree Control System Functions

This chapter discusses the system functions unique to the tree control.

Bulk Tree Load

Use this system function to load the entire contents of the tree control from cache.

Parameters

Tree Control

Input, required. The tree control form control (FC) to affect.

Cache Name

Input, required. The name of the cache from which to acquire data for the contents of the tree. Set the parameter to an applicable object from the object list.

Data Structure

Input, required. The name of the data structure used to organize the cache. Double-click <Get Structure Name> to select the structure to use.

Node Value Column

Input, required. The data to use for identification. This value is not a true ID in the traditional sense; in this system function, it is used primarily for matching with values in the Parent Value Column to determine parent/child relationships. If you want to use node value as a true identifier, ensure that each row will have a unique value in this column (many tree control system functions use node value as an ID). Set the parameter to an applicable object from the object list.

Node Description Column

Input, required. The label to display for each node. Set the parameter to an applicable object from the object list.

Parent Value Column

Input, required. The data to use to determine parent/child relationships. This value indicates which row is the parent of the current row. When runtime constructs the tree, it will place the current row as a child of the row with the corresponding value in Node Value Column. Set the parameter to an applicable object from the object list.

Root Node Value

Input, required. The value that indicates the root node. Ultimately, the uppermost node of the tree, the root node, has no parent. Runtime compares this value with the values in the Node Column Value parameter. The row with the matching value becomes the root node.

Contract Tree Node

Use this system function to collapse a given node.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node to collapse. Set the parameter to: <Currently Selected Node>, <Last Inserted Node>, or <Currently Expanding/Collapsing Node>. [Expand Tree Node](#)

Delete Tree Node

Use this system function to delete a node (and its children) from the tree. You can also delete just the children of a node or all of the nodes in the tree.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node to delete. Set the parameter to: <Currently Selected Node>, <Last Inserted Node>, <All Child Nodes Under the Current Node>, <All Nodes>, <With Specified Node Value>. Use the last option in conjunction with the Node Value parameter.

Node Value

Input, optional. The node to delete. This parameter is required only if you set Node to <With Specified Node Value>. Set the parameter to an applicable object from the object list.

Expand Tree Node

Use this system function to expand a node (provided that the node is expandable).

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node to expand. Set the parameter to: <Currently Selected Node>, or <Last Inserted Node>. [Contract Tree Node](#)

Get Node Information

Use this system function to acquire the display text of a node and its associated node value

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node about which to acquire information. Set the parameter to: <Currently Selected Node>, <Last Inserted Node>, or <Currently Expanding/Collapsing Node>.

Node Text

Output, required. The object to which to return the text displayed with the node. The data type of this parameter should be the same as that is displayed by the node. For example, if the node is displaying a numeric value, this parameter should be a numeric value. Set the parameter to an applicable object from the object list.

Node Value

Output, required. The object to which to return the value of the node. The data type of this parameter should be the same as that of the value currently stored in the node. For example, if the node currently has a numeric value, this parameter should be a numeric value. Set the parameter to an applicable object from the object list.

Additional Notes

Store the node value while inserting the node or by using the **Set Node Information** system function. A typical use is to store a key value associated with the tree node. Data of any type can be stored in the node value as long as you maintain consistency in storing and retrieving the value.

Returns

This system function returns the text associated with the node to the object indicated by Node Text, and the node value to the object indicated by Node Value.

Get Node Level

Use this system function to determine the vertical level of a node in the tree. The root node is 0, so its first child is at a level of 1, its second at 2, and so on.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node about which to acquire information. Set the parameter to: <Currently Selected Node>, <Currently Expanding/Collapsing Node>, or <Currently Deleted Node>.

Return To

Output (numeric), required. The object to which to return the node level. Set the parameter to an applicable object from the object list.

Returns

This system function returns the level of the node to the object indicated by Return To.

Get Tree Node Handle

Use this system function to acquire the table handle associated with a particular tree node.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node for which to acquire the handle. Set the parameter to: <Currently Selected Node>, <Last Inserted Node>, <Root Node>, <Currently Expanding/Collapsing Node>, or <Currently Deleted Node>.

Index

Input, required. The index of the handle. Applications can associate more than one handle to one tree node. Use this parameter to differentiate multiple handles for the same node. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Handle

Output, required. The object to which to return the handle associated with the node. Set the parameter to an applicable object from the object list.

Returns

This system function returns the handle associated with the identified tree node to the object specified by Handle.

Insert Tree Node

Use this system function to insert a node into the tree.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The position in the tree where you want to insert the node, relative to this node. Set the parameter to insert the node as a child of the current node (<Under Currently Selected Node>), to insert the node as a sibling of the current node (<After Currently Selected Node>), <After Last Inserted Node>, <Under Root Node>, <Under Currently Expanding Node>, or <Under Specified Parent Value>. Use the last option in conjunction with the Parent Value parameter.

Node Text

Input, required. The object to which to return the text displayed with the node. The data type of this parameter should be the same as that is displayed by the node. For example, if the node is displaying a numeric value, this parameter should be a numeric value. Set the parameter to an applicable object from the object list.

Node Value

Input, required. The object to which to return the value of the node. The data type of this parameter should be the same as that of the value currently stored in the node. For example, if the node currently has a numeric value, this parameter should be a numeric value. Set the parameter to an applicable object from the object list.

Expandable?

Input, required. An indicator of whether the user can expand and collapse the node. Set the parameter to <Yes> or <No>.

Parent Value

Input, optional. The ID of the node you want to become the parent of the node you are inserting. Use this parameter if you set Node to <Under Specified Parent Value>. Set the parameter to an applicable object from the object list.

Set Bitmap Scheme

Use this system function to set the default bitmap schema that will be used by the tree nodes in the tree control.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Open Bitmap

Input, required. The bitmap to display for a node when it has been expanded. Set the parameter to a particular bitmap (double-click <Choose Tree Bitmap> to browse for one) or the standard bitmap (<Default>).

Closed Bitmap

Input, required. The bitmap to display for a node when it has been collapsed. Set the parameter to a particular bitmap (double-click <Choose Tree Bitmap> to browse for one) or the standard bitmap (<Default>).

Leaf Bitmap

Input, required. The bitmap to display for a leaf node with no children. Set the parameter to a particular bitmap (double-click <Choose Tree Bitmap> to browse for one) or the standard bitmap (<Default>).

Additional Notes

The bitmaps must be of size 16 x 16 and should reside in the treebmps subdirectory of the resource directory (for example: d:\b7\appl_pgf\res\treebmps). If a bitmap fails to load, the application uses the corresponding standard bitmap. The ideal place for using this system function is during on event, **Dialog Is Initialized**.

Set Node Bitmap

Use this system function to set a bitmap for a particular node, no matter its state.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node for which to set the bitmap. Set the parameter to:
<Currently Selected Node>, **<Last Inserted Node>**, or **<Currently Expanding/Collapsing Node>**.

Bitmap

Input, required. The bitmap to display for the node. Set the parameter to a particular bitmap (double-click **<Choose Tree Bitmap>** to browse for one) or the standard bitmap (**<Default>**).

Additional Notes

After the bitmap for a tree node is set using this system function, the default scheme changes for that node. The system does not apply the expanded and collapsed bitmaps to the node.

The bitmaps must be of size 16 x 16 and should reside in the treebmps subdirectory of the resource directory (for example: d:\b7\appl_pgf\res\treebmps). If a bitmap fails to load, the application uses the corresponding standard bitmap. The ideal place for using this system function is on the event **Dialog Is Initialized**.

Set Node Information

Use this system function to change the node value and the text displayed for the node.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node to affect. Set the parameter to: <Currently Selected Node>, <Last Inserted Node>, or <Currently Expanding/Collapsing Node>.

Node Text

Input, required. The text to display for the node. Set the parameter to an applicable object from the object list.

Node Value

Input, required. The value to apply to the node. This value is often used for identification purposes, so you might want to ensure that it is unique. Set the parameter to an applicable object from the object list.

Additional Notes

A typical use of a node value is to store a key value associated with the tree node. Data of any type can be stored in the node value as long as you maintain consistency in storing and retrieving the value.

Set Node Text

Use this system function to apply or change the text associated with a given node. This is the text that users can see.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node Text

Input, required. The text to apply to the node for display. Set the parameter to an applicable object from the object list.

Node Value

Input, required. The value of the node to affect. Set the parameter to <Root Node> or an applicable object from the object list.

Set Tree Node Handle

Use this system function to associate a handle with a particular tree node.

Parameters

Tree Control

Input, required. The tree control FC to affect.

Node

Input, required. The node to which to associate the handle. Set the parameter to:
<Currently Selected Node>, **<Last Inserted Node>**, **<Root Node>**, or **<Currently Expanding/Collapsing Node>**.

Index

Input, required. The index of the handle. Applications can associate more than one handle to one tree node. Use this parameter to differentiate multiple handles for the same node. Set the parameter to an alphanumeric constant (**<Literal>**), **<Blank>**, **<Zero>**, or an applicable object from the object list.

Handle

Input, required. The object to which to return the handle associated with the node. Set the parameter to an applicable object from the object list.

34

Understanding Wizard Controls

This chapter contains the following topics:

- [Section 34.1, "Wizard Controls"](#)
- [Section 34.2, "Wizard Control Design-Time Considerations"](#)
- [Section 34.3, "Wizard Control Events"](#)
- [Section 34.4, "Wizard Control Runtime Processing"](#)
- [Section 34.5, "Wizard Control Transaction Processing"](#)
- [Section 34.6, "Wizard Control System Functions"](#)

34.1 Wizard Controls

The *wizard control* is the primary component of the wizard. It can reside only on the wizard form type, and only one wizard control can exist on any given wizard form. Each wizard control contains multiple *pages*; each page is comprised of one *subform* or *alias*. Each page corresponds to a task that the user must complete to finish the wizard. During runtime, the wizard displays the first page and waits for the user to complete the task. Typically, you use the pages to prompt the user to provide input. After completing the task, the user clicks the Next button, and the wizard validates page one data and displays the second page in the list (index). In a standard scenario, the user continues in this fashion until the last page is reached, at which point the user clicks Finish and the wizard re-validates and commits the data before terminating.

You can opt to display the *progress list* itself (which also enables the user to jump between tasks by clicking a task in the progress list), and you can also display a progress indicator which shows how much of the wizard is complete without listing the individual tasks. The progress list indicates which pages have been completed, which ones have errors, and which ones have warnings by displaying icons next to affected tasks. You cannot alter these icons programmatically. Additionally, the control automatically provides certain buttons on each page of the wizard. It provides a Previous button on all but the first page, a Next button on all but the last page, a Finish button on the last page, and an Exit button on all pages. You can add additional buttons such as Save and Cancel, but you cannot remove these default buttons.

You can enable users to start a wizard and then save it to finish later. If a user saves, the control does not automate saving the data for you. Instead, you must react to the **WIZARD:Save for Re-entry** event and save the data programmatically. If you enable revisits on the wizard, the interface provides a Save For Later button as well.

The subforms included in the wizard have no form- or control-based limitations; that is, you can employ the full range of subform features and system functions. Therefore, to create an effective wizard application, you must be proficient with subforms.

Each subform is parented to the wizard form which means that subforms cannot communicate with each other directly; they must share data using the wizard form. Data can be passed in one of three ways: from page to wizard form (child to parent), from wizard form to page (parent to child), or in both directions. Only data in the child page data structure can be passed to the parent wizard form. Consequently, you must plan the data structures carefully. In general, objects in the data structure will fall into one of two categories: items which the page will want to share with other pages, and items which the page will require such as input from other pages or the wizard form itself.

You can embed subforms and use an alias to a reusable subform. You can programmatically reorder the pages (therefore reordering the tasks). You can also indicate a starting page other than the first one in the list. As the user progresses through the wizard, tasks that the user has completed are referred to as *upstream tasks*. Those which the user has yet to complete are referred to as *downstream tasks*.

You are not limited to forms inside the wizard. You can use form interconnections to link to forms outside of the wizard. If you use an interconnection, the forms that appear are called *satellite pages*. Like the wizard itself, satellite pages expand to fill the entire frame; the standard JD Edwards EnterpriseOne navigation menu is hidden. When the user clicks OK on the satellite page, the wizard reappears displaying the page containing the object that triggered the interconnection.

Forms have different statuses including indeterminate (no state), complete, and incomplete (but continue). No state is the initial, pristine state of a page and it indicates that the page has been initialized but not yet visited. Complete indicates that the page has been visited, finished successfully, and validated. Incomplete indicates that the page has been visited but not yet finished or validated. All pages in the wizard must be at a status of complete before runtime can commit data.

Runtime validates all wizard pages when the user clicks Finish, even hidden and disabled pages. To prevent a page from being validated and saved, enable the Form Design Aid (FDA) property, Suppress Validation and Save. During runtime, use the system function, **Suppress Wizard Page Validation and Save**. As a rule of thumb, call the **Suppress Wizard Page Validation and Save** system function for all hidden and disabled pages in the wizard to prevent Finish button processing on them.

When a wizard application is launched, runtime sets the status of all pages to no state as part of the initialization process. When the user enters a page of indeterminate status, runtime changes its status to incomplete to reflect the fact that the user has visited the page. These are the only times that runtime sets the page status.

Application logic is expected to manage page status otherwise. However, runtime will not permit the application to assign a complete status to a page until all its errors have been resolved.

All pages must be at a status of complete for the Finish button process to succeed, even hidden and disabled pages, unless validation is suppressed for those pages. If any page is not set to complete, the Finish button process will fail and no data will be saved. To set page status, use the **Set Wizard Page Status** system function.

See Also:

- [Using Subforms and Subform Aliases](#).
- [Understanding Wizard Forms](#).

34.2 Wizard Control Design-Time Considerations

The wizard control permits the standard control property settings; likewise, the standard options are available for subforms and forms as well. It is recommended that you always enable transactions for all objects in the wizard, however. Since the runtime engine gathers and saves all data and then commits it in a single transaction if the wizard finishes with no errors, disabling transactions on any wizard object is counterproductive.

In addition to the standard control property values, the wizard control includes these wizard control-specific values as well:

- Progress Indicator

If you want to show how much of the wizard has been completed, you can choose to do so as a percentage value or as the number of tasks completed out of the whole (X of Y).

- Enable Re-entry Save

When enabled, this option displays the Save For Later button on every page. You can add logic to the button that enables users to stop using the wizard before completing it, save their work up to that point, and then return to the wizard later to finish. Because the wizard hides the JD Edwards EnterpriseOne navigation menu, you might consider enabling this option for lengthy wizards. Users cannot launch other JD Edwards EnterpriseOne applications if they cannot access the JD Edwards EnterpriseOne navigation menu.

Note: Enabling this option only displays the Save For Later button. You must add application logic to save data and then load it and return to the correct page when the user returns. The wizard will not commit data to the database until the entire wizard is completed.

- Enable Progress List

When enabled, this option displays the list of tasks (each task corresponds to a wizard page) comprising the wizard. As the user completes the tasks, his or her advancement is indicated in the progress list.

Users can jump to up- or downstream tasks by choosing a task in the list. The user cannot jump to disabled, hidden, or unvisited pages using the progress list.

- Suppress Validation and Save

This option is applied on individual subforms. Typically, you apply it to subforms which you have also disabled or hidden. You can use this option to prevent such pages from being processed at runtime, thereby making the wizard more efficient. Furthermore, the Finish button process requires that every validated page be at a status of complete. Depending on how you design the application, it might not be possible for disabled and hidden pages to achieve this status. Therefore, you must suppress validation and save on such pages or the wizard application will never commit data to the database.

The ability to suppress validation and save functions can be enabled and disabled during runtime with the **Suppress Wizard Page Validation and Save** system function.

34.2.1 Implementing Re-entry Save

When you select the Enable Re-entry Save option, the application must perform data saves and any other functions you want the application to provide if a user decides to save the wizard and return later.

This list describes a method for implementing re-entry save when each wizard page contains a separate business view:

1. Create a working table for each page.
2. On the **Wizard:Save For Reentry** event of each wizard page, save the data into the working tables.
3. When the user reenters, on the **Wizard:Subform is Initialized** event of each wizard page, load data from the working table and assign the values to form controls and variables.

This event occurs after the system applies next numbers and default values, so they will not override the user's earlier data.

This list describes a method for implementing re-entry save when each wizard page shares the same business view or has no business view:

1. Have the wizard control collect data from each page and store it in event rule (ER) variables.
The mapping links between the wizard pages and the wizard form must be bidirectional so that the wizard form can push data into each page.
2. On the **Wizard is Finished** event, have the wizard control save the data using table I/O or business functions.
3. Create one working table to hold reentry data.
4. On the **Wizard:Save For Reentry** event of each wizard page, pass the data from the child subform so the parent wizard form can save the data into the working table using table I/O or business functions.
5. When the user reenters, on the **Post Wizard is Initialized** event, have the wizard control load data from the working table and into ER variables.

Setting ER variables on this event enables you to override the next numbers applied by the system.

6. On the **Wizard:Subform is Entered** event, the system automatically populates subform interconnect (SI) variables from the parent wizard form. The page must assign the SI variables to form controls to display them.

34.3 Wizard Control Events

Wizard forms and their subforms have these wizard form-specific events (some occur on the wizard form and some on its subforms), and they typically occur roughly in this order:

- **Wizard is Initialized**

This form-level event indicates that the form and control have been created, security has been applied, system variables have been initialized, and form data structures have been loaded. It is the only event on which you should use the **Set Wizard Form Mode** system function (to change the form mode to add, update, or copy), and the only one of two events on which you should use **Set Selected Wizard Page** (to start at a page other than the first visible, enabled page in the

index).

Wizard is Initialized is also where you should hide, show, disable, or enable a page, or suppress page processing for a page, as appropriate. Finally, this is also when you should rearrange the page order if necessary (**Set Wizard Page Index**).

- **Post Wizard is Initialized**

This form-level event fires immediately after **Wizard is Initialized**, indicating that the initialization process is complete and runtime is poised to enter the first page in the wizard. Any logic that should be applied to the **Wizard is Initialized** event can be applied here also. You can use the **Set Selected Wizard Page** system function on this event, but not after; runtime determines which page is the "first page" to enter based in part on whether you set a selected page.

- **WIZARD:Subform is Initialized**

This subform-level event indicates that a subform has been initialized (including the standard processing for subforms in add mode). The event fires for each subform in the wizard control, in order. This event is called only once during the life cycle of a wizard form per session. If the user worked on the wizard previously and saved (that is, if this is a reentry), this event can be used to load previous session data and then assign the data to form controls and ER variables.

- **WIZARD:Subform is Entered**

This subform-level event indicates that the page status is incomplete, that parent mapping links have been reestablished, and that the SI values have been updated. The event fires every time a user visits a page by accessing it from a previous page. The SI values should contain all current information. Assign SI values to form controls and grid rows if necessary. Set filter values and QBEs at this point if the Automatically Find On Entry option is enabled.

- **WIZARD:Post Subform is Entered**

This subform-level event indicates that any processing required by the Automatically Find on Entry option is complete. This event is called every time a user visits this page from an earlier page. This is called after the **Subform Entered** event and after the automatic data fetch if Automatically Find On Entry is enabled.

- **WIZARD:Save for Re-entry**

This subform-level event fires on every page, except those on which **Suppress Validation and Save** is enabled, when the user clicks the Save for Re-entry button. the application should save the data on the current page to a temporary working table on this event.

- **WIZARD:Validate Subform**

This subform-level event occurs before runtime conducts customized validation of the form controls and grid rows. Runtime always performs the data dictionary validations unless the Suppress Validation and Save option is enabled for the page. At this point, the application code should set the page status to complete by calling the system function **Set Wizard Page Status**.

- **WIZARD:Subform is Exited**

This subform-level event fires once for each subform in the control after the transactions have been processed and runtime is preparing to close the wizard form. It is the last subform-level event to fire for each subform before the wizard closes. On this event, the application should copy form controls or grid rows to SI variables, then call the **Update Parent** subform system function to update the wizard form. You can perform this call conditionally, such as when the user clicks

Next or Finish.

- **Page is Exited - Before**

This form-level event indicates that subform validation is complete. It is the last event to fire before runtime exits the current subform.

- **Page is Exited - After**

This form-level event fires immediately after any logic on **Page is Exited - Before** finishes processing.

- **Wizard is Finished - Before**

This form-level event indicates that page-level validation is complete and that runtime is about to begin passing data from the wizard to the business view.

- **Wizard is Finished - After**

This form-level event fires after the business view is loaded with data from the wizard form but before the data is committed to the database. It is on this event that you should code any data operation that needs to be included in the transaction. Do not trigger a form interconnect on this event.

- **Wizard is Exited**

This form-level event fires before form close; it is the last event to fire. This is the point at which you should add code to clean up caches and other resources used by the wizard. If you require a form interconnect (such as for a confirmation form), this event occurs outside the transaction boundary and is therefore safe to use to trigger the interconnection.

Note: Even though wizard control may resemble a tab control, none of the tab page events are fired for wizard pages. Wizard pages are not tab pages.

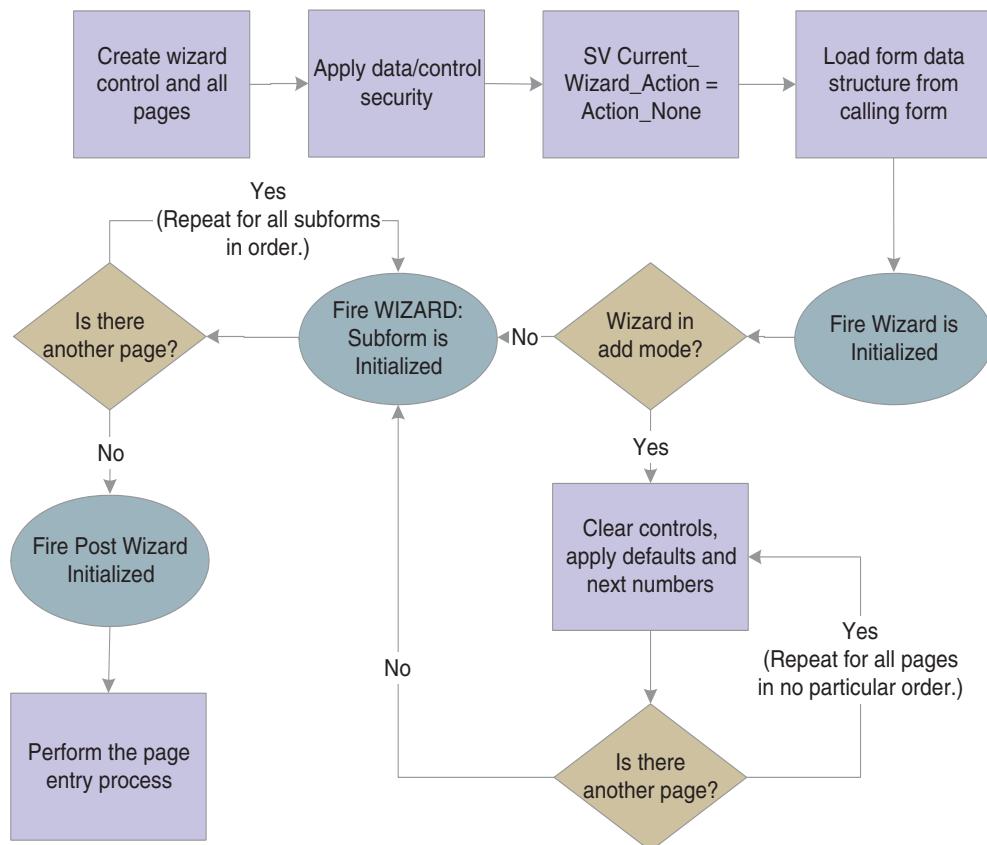
34.4 Wizard Control Runtime Processing

This section discusses the runtime processing of the wizard control.

34.4.1 Initialization

Form initialization is the only point at which you can change the form mode (add, edit, or update), and you should do so with **Set Wizard Form Mode** on the **Wizard is Initialized** event because runtime checks the form mode immediately after firing **Wizard is Initialized**.

This flowchart illustrates how runtime initializes the wizard form:

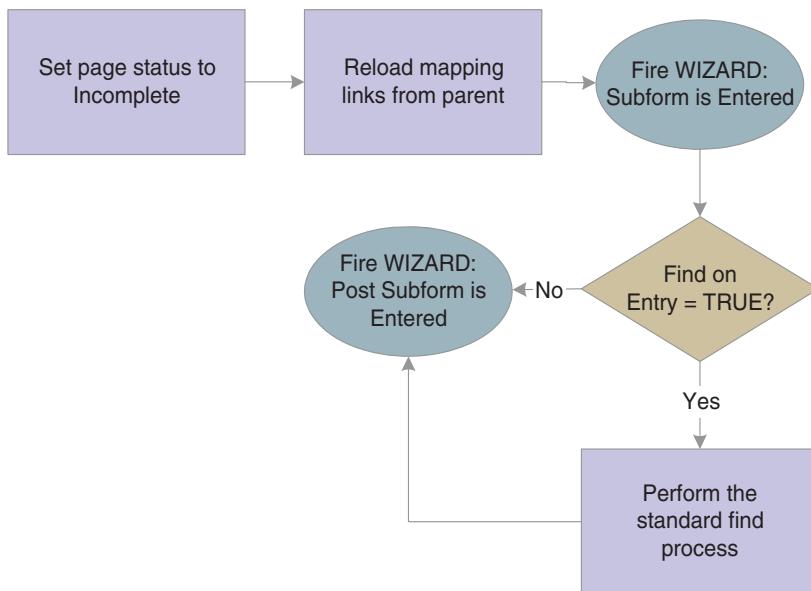
Figure 34–1 Wizard form initialization

34.4.2 Page Entry

After the wizard form is initialized, runtime enters the first page in the wizard control. Runtime determines the first page to be the first enabled and visible page it encounters, starting from either the first page in the index or the selected page indicated by the **Set Selected Wizard Page** system function.

When the user enters a new page, the system automatically populates the SI variables from the parent form variables before **Wizard:Subform is Entered** fires. You can place logic to populate subform fields with this data on either event that fires during page entry.

This flowchart illustrates how runtime processes page entry:

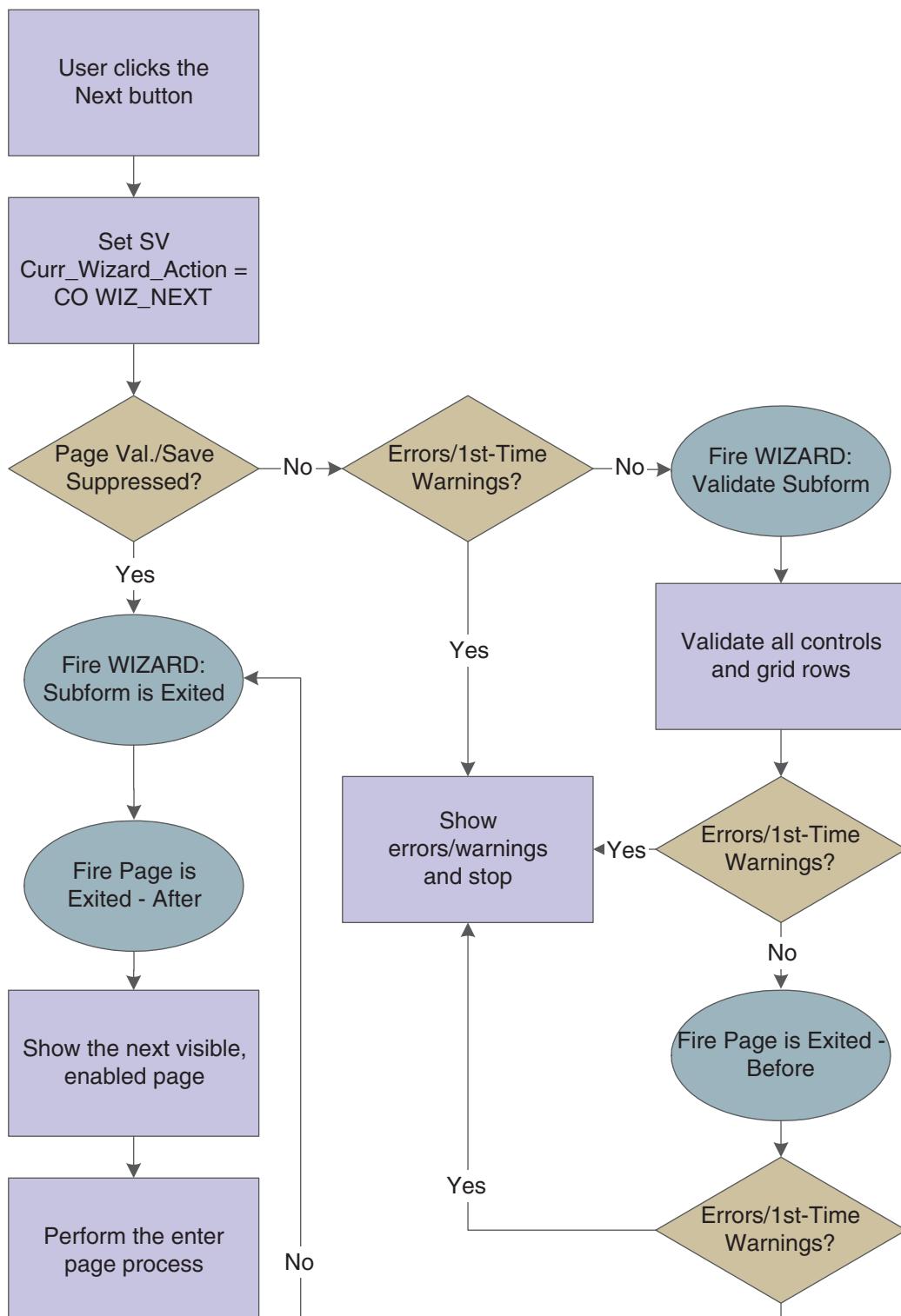
Figure 34–2 Wizard control page entry processing

34.4.3 Next Button Processing

The Next button causes runtime to validate the current page. If no errors occur due to validation, then runtime displays the next downstream, visible, enabled page in the index. If errors occur, runtime does not display the next page; instead, it highlights the errors so that the user can fix them before moving forward.

When the user exits a page, the system does not automatically populate the SI variables to the parent form variables. Therefore, the application must assign appropriate SI variables and call the **Update Parent** subform system function in the **WIZARD:Subform is Exited** event, if necessary.

This flowchart illustrates the runtime processing that occurs on the current page when the user clicks the Next button:

Figure 34–3 Wizard control Next button processing

34.4.4 Previous Button Processing

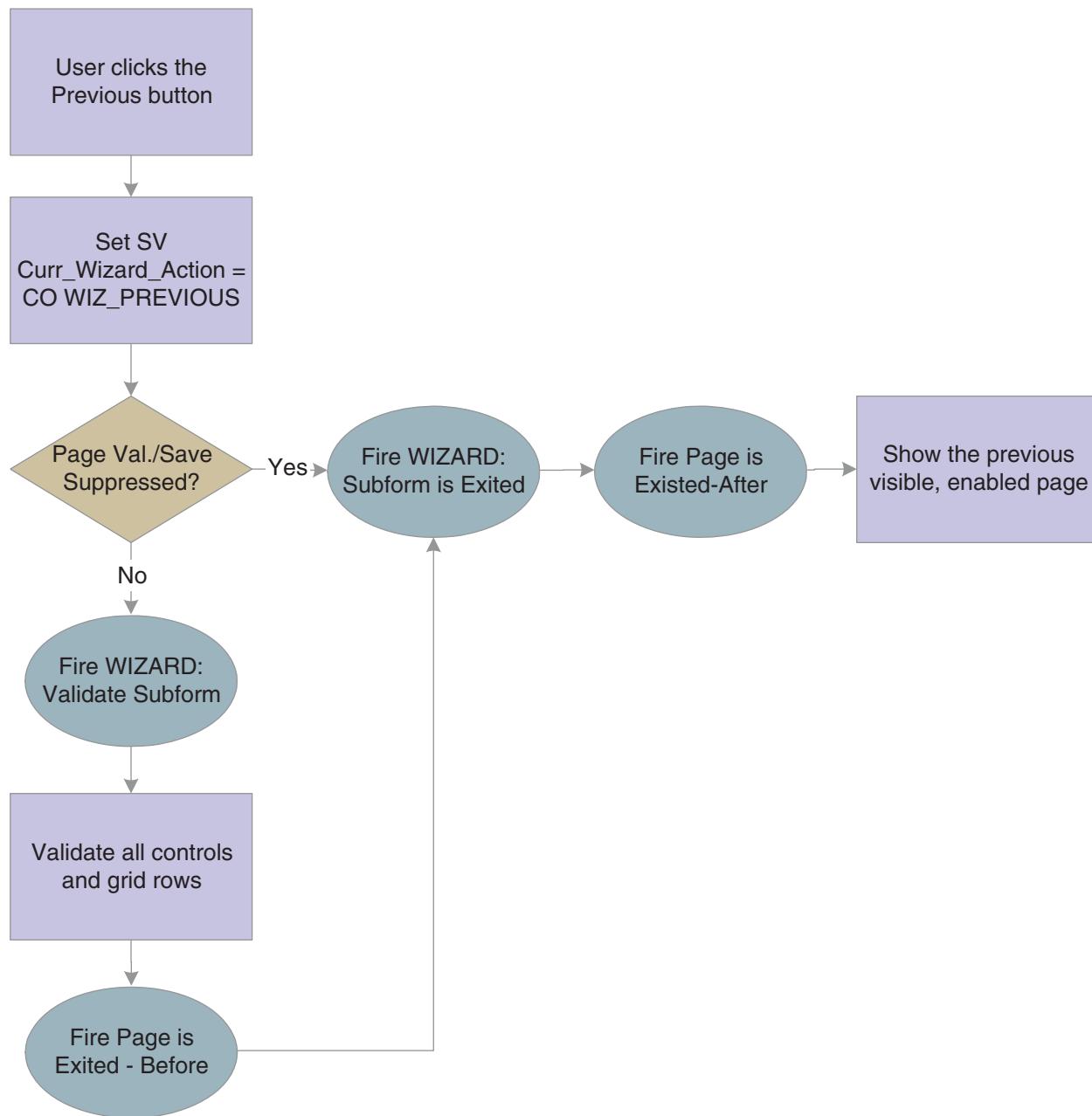
When the user clicks the Previous button, runtime displays the previous upstream, visible, enabled page in the index, including the data entered by the user. To prevent

the data from being cleared, runtime does not run the enter page process. If errors occur during validation, runtime marks the pages with errors in the progress list (if visible), but it always displays the previous page, nonetheless.

If the user wants to change previously entered data, he or she may do so, but must click Next to save the changes.

This flowchart illustrates the runtime processing that occurs when the user clicks the Previous button:

Figure 34–4 Wizard control Previous button processing



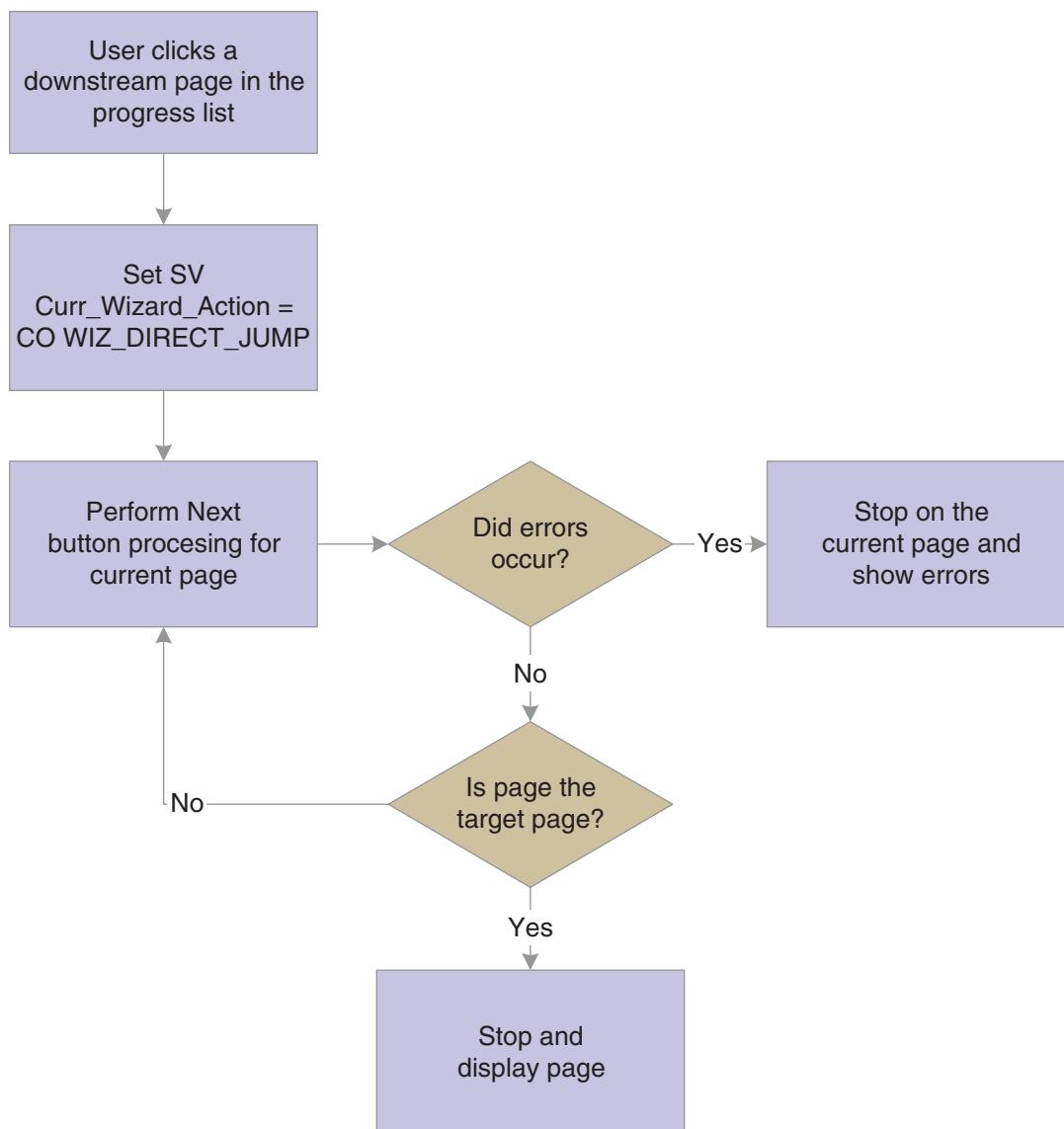
34.4.5 Jumping Up- and Downstream

When the progress list is enabled, users can access previously-visited pages directly, effectively skipping one or more tasks if they select a page that is not immediately before or after the current page in the index.

When a user jumps downstream, runtime loops through each page between the start and target inclusively, applying Next button processing which includes error processing. If a page between the starting page and the target page throws an error in the course of Next button processing, runtime halts processing and displays the page on which the error occurred. If the user successfully jumps to the target page, runtime performs the page entry process.

This flowchart illustrates the runtime processing that occurs when the user tries to jump to a downstream page:

Figure 34–5 Wizard control downstream jump processing



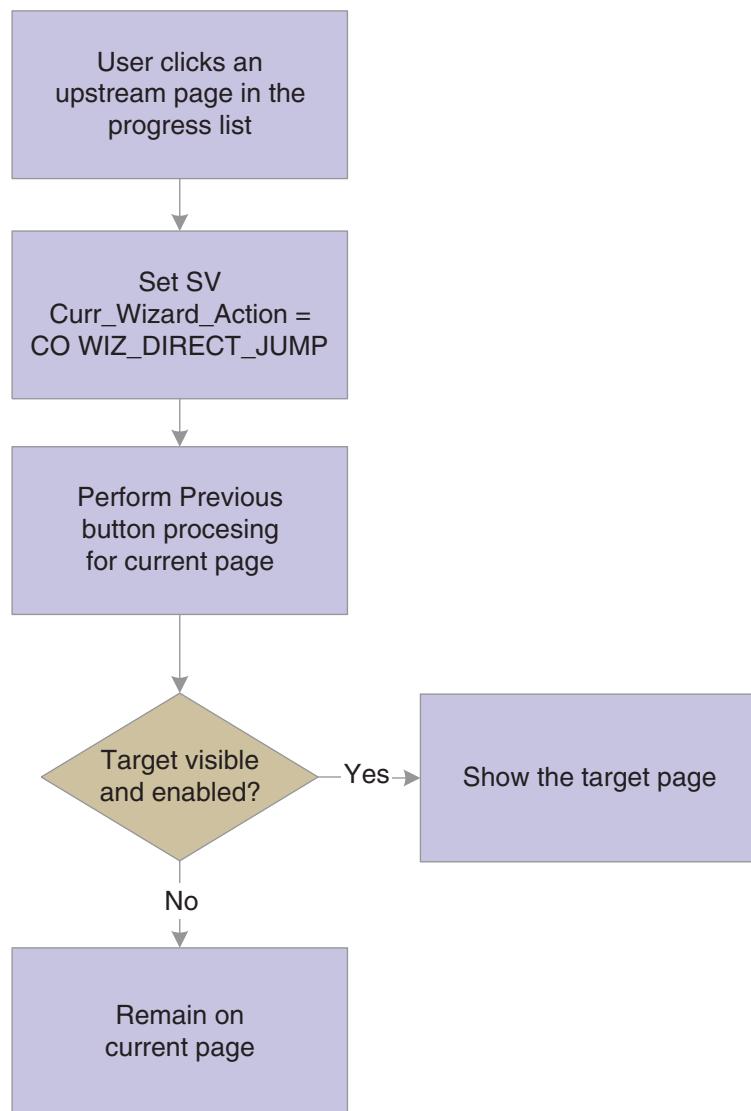
When a user jumps upstream, runtime performs Previous button processing on the current page and then displays the target page as long as that page is visible and

enabled. If the processing produces errors, runtime displays them, but it does not prevent the display of the target page. Runtime does not process the pages between the current page and the destination page because it is assumed that upstream pages should not be affected by downstream pages. Furthermore, runtime does not run the page entry process because nothing should change on the destination page.

If the user wants to change previously entered data, he or she may do so, but must click Next to save the changes.

This flowchart illustrates the runtime processing that occurs when the user tries to jump to an upstream page:

Figure 34–6 Wizard control upstream jump processing

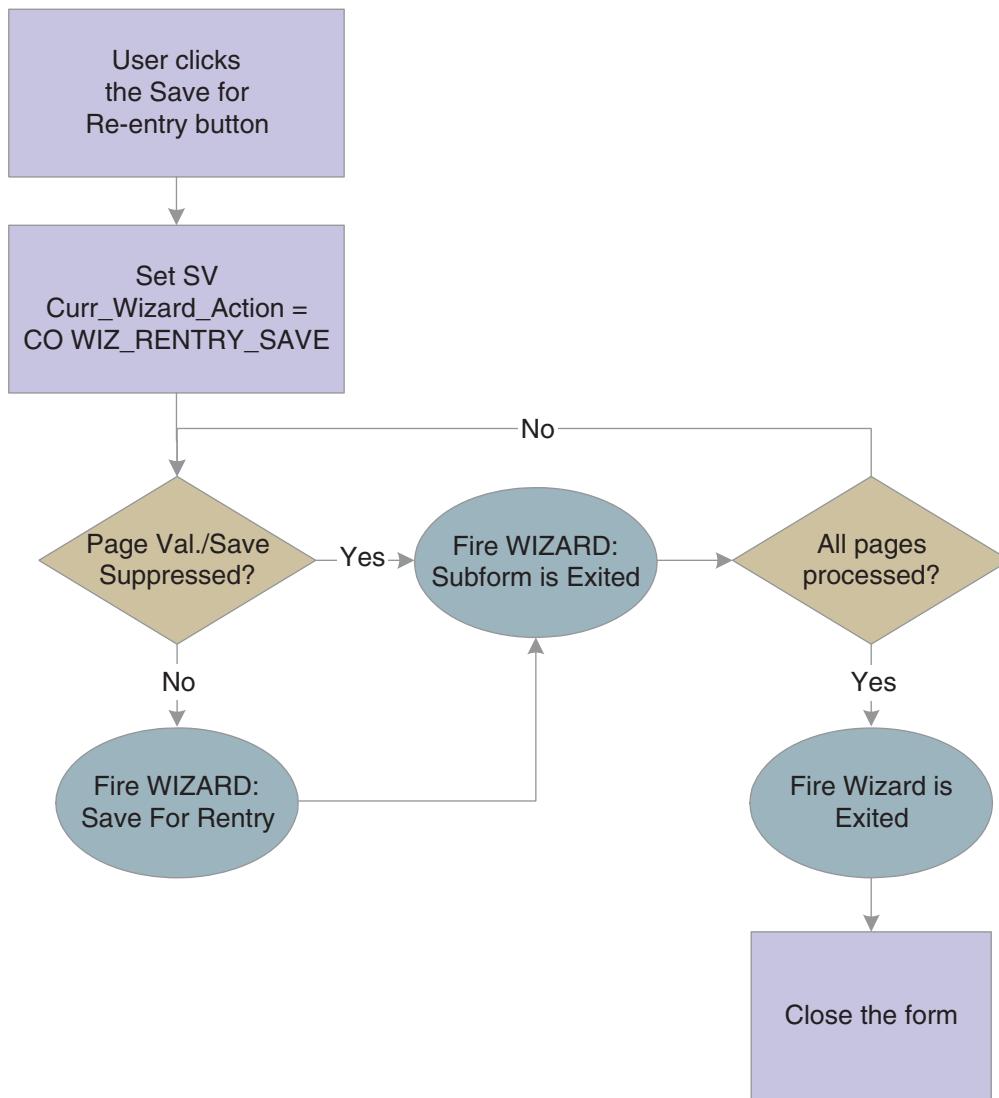


34.4.6 Save for Re-entry Button Processing

If you enable it, the user can click the Save for Re-entry button to save the data to that point and close the form. You are responsible for creating logic to perform the data save (and to enable the user to return to the correct spot later).

This flowchart illustrates the runtime processing that occurs when the user clicks the Save for Re-entry button:

Figure 34–7 Wizard control Save for Re-Entry button processing



34.4.7 Cancel Button Processing

When the user clicks Cancel, runtime confirms that the user actually wants to quit without saving any changes. If the cancel is confirmed, runtime sets SV Curr_Wizard_Action = CO WIZ_CANCEL, fires **Wizard:Subform is Exited** for every page in the wizard, followed by **Wizard is Exited**. Then runtime closes the wizard form.

34.4.8 Finish Button Processing

Finally, runtime is ready to perform transaction processing. Runtime processes each subform in order, writing the data to the business view, but not committing to the database. If a subform has a save button that was clicked, runtime performs standard save button processing. After all of the pages have been processed, runtime commits all database changes in one transaction and then proceeds to close each subform and then the wizard form.

This three-part flowchart illustrates the runtime processing that occurs after the successful validation of all pages in the wizard:

Figure 34–8 Wizard form Finish button processing, part 1 of 3

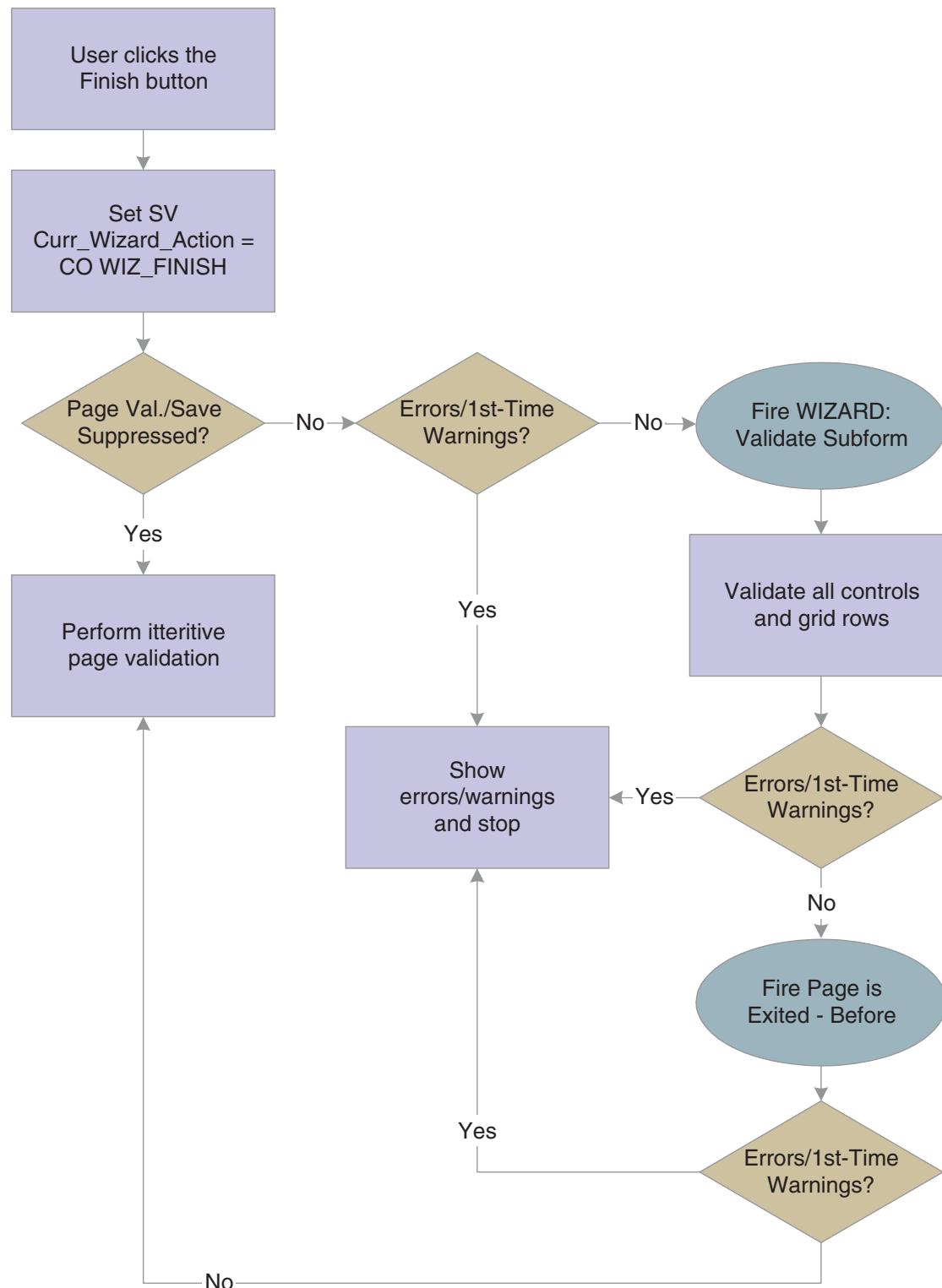


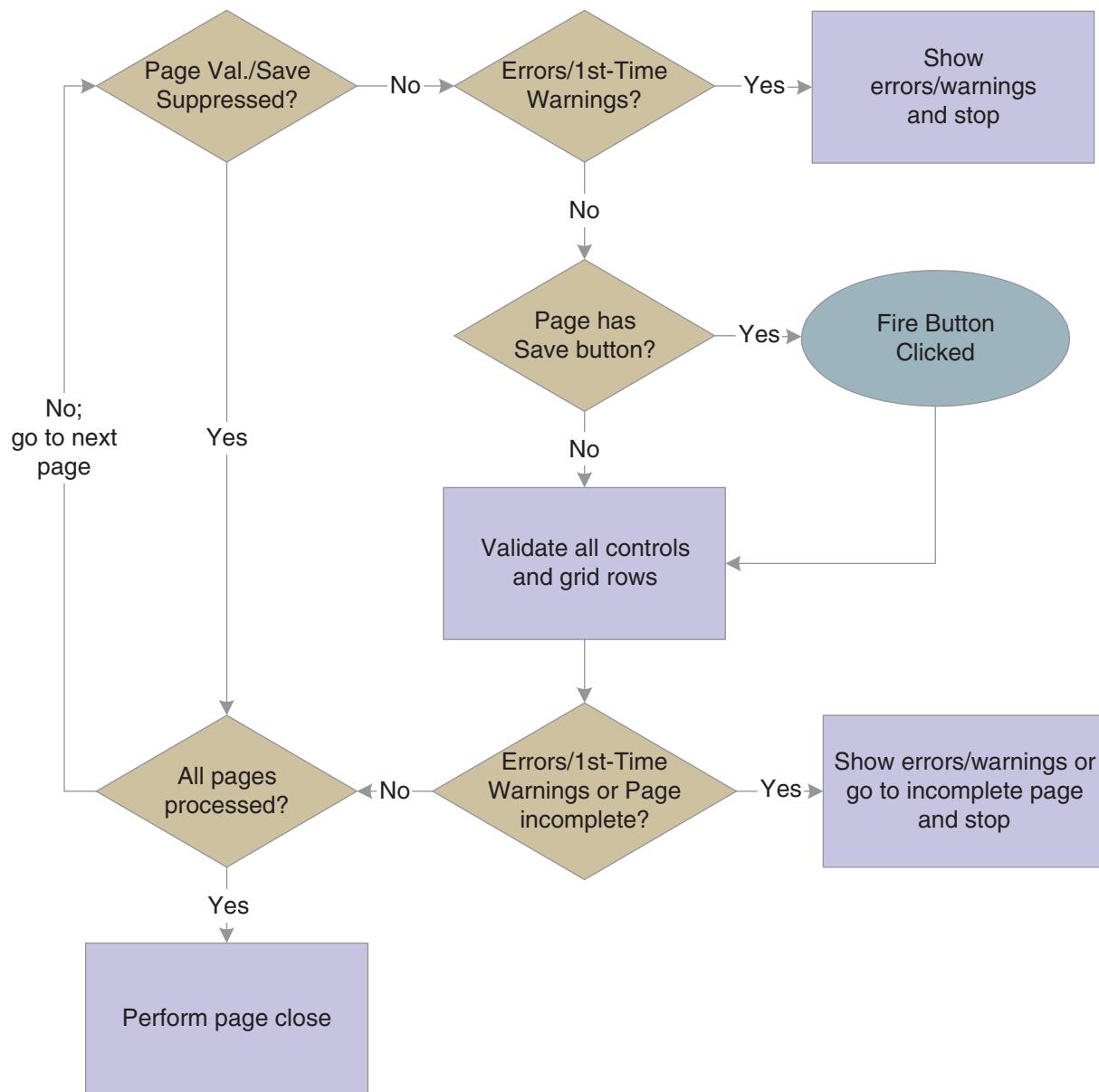
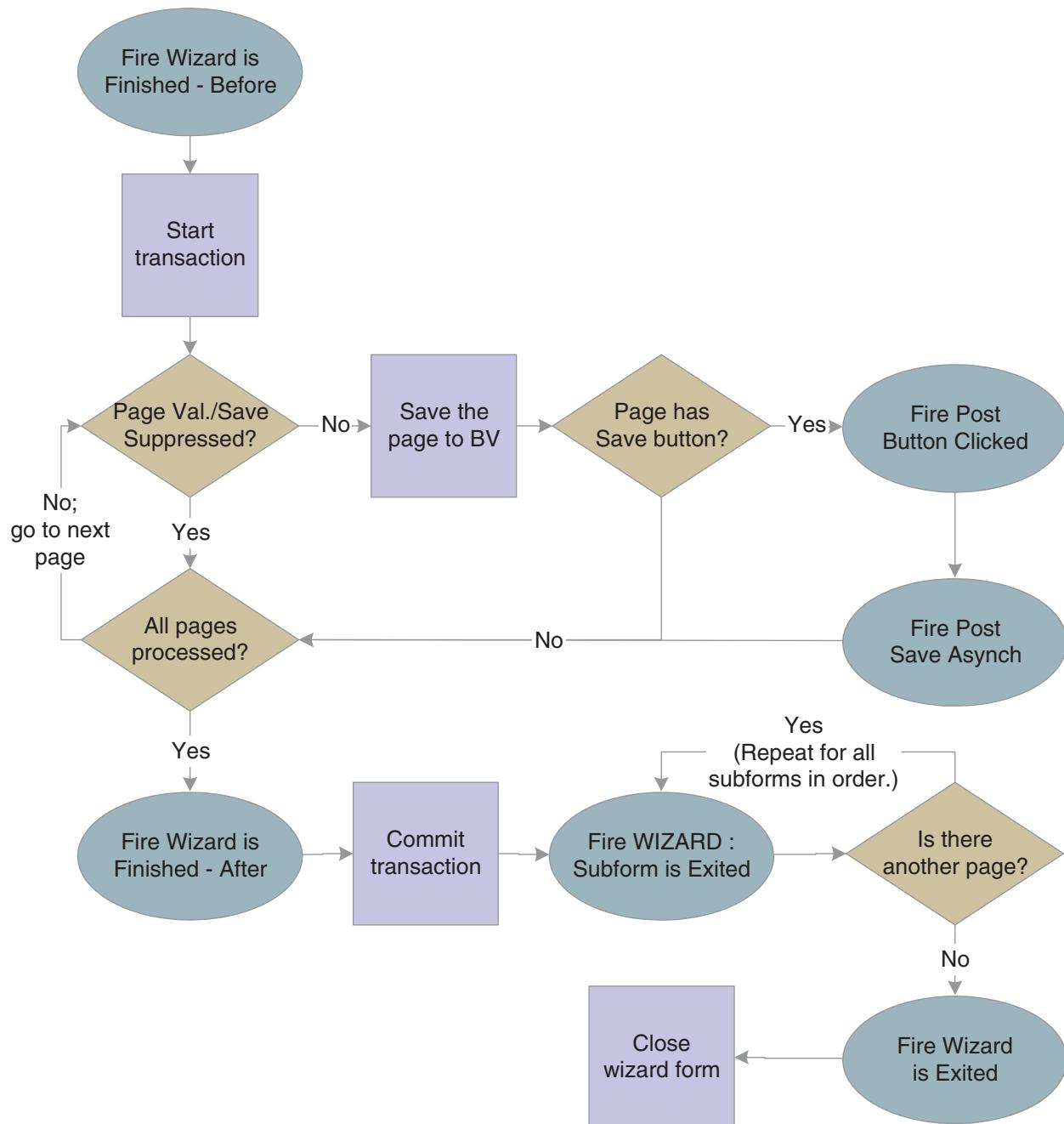
Figure 34–9 Wizard control Finish button processing, part 2 of 3

Figure 34–10 Wizard control Finish button processing, part 3 of 3

34.5 Wizard Control Transaction Processing

All data is saved with a single transaction when the user clicks the Finish button. Contrast this behavior with clicking the Next button which validates data, but does not save it to the business view. Consequently, it is recommended that you limit the number of pages in a wizard for performance purposes.

The save transaction boundary is composed of and defined by this sequence of events:

1. Runtime initiates the transaction.
2. Runtime saves each page into the business view.

3. Runtime fires the **Post Button Clicked** event if it encounters a Save button on a page (the event fires for each button encountered).
4. Runtime fires the **Post Button Async** event for each Save button instance in a separate thread.
5. Runtime fires the wizard control event: **Wizard is Finished-After**.
6. Runtime performs the commit/rollback transaction.

Do *not* call form interconnects during any of the events inside the transaction boundary. Opening a new form will prolong the transaction and severely impact system performance.

For this reason, do not launch a separate "confirmation form" from any of these events. Instead, it is recommended that you use the last page of the wizard as the confirmation page. If it must be a separate form, then launch a confirmation form in the **Wizard is Exited** event. You can use the system variable SV_Curr_Wizard_Action and launch only the confirmation form when the user clicks the Finish button.

34.6 Wizard Control System Functions

You can use the **Disable Subform** and **Enable Subform** system functions to hide and show pages. Because the wizard form is the parent of all subforms, only the wizard form can disable, enable, hide, or show a given page. This section lists the system functions that are specific to the wizard form and the wizard control.

Get Current Wizard Page ID

Use this system function to get the ID of the current page in the wizard. (The control ID of a subform can be found in the properties for the subform in FDA.)

Parameters

Wizard Control

The wizard form control (FC) to affect.

Wizard Page ID

Input (integer), required. The object to which to assign the return value that indicates the ID of the current page. Set the parameter to an applicable object from the object list

Returns

The system function returns the ID of the current wizard page to the object indicated by Wizard Page ID.

Get Wizard Page Index

Use this system function to retrieve the index of a selected wizard page in relationship to the other wizard pages. The list of wizard pages starts with an index of one.

Parameters

Wizard Control

The wizard FC to affect.

Subform

Input, required. The page for which you want the index value. Set the parameter to an applicable object from the object list.

Index

Input (integer), required. The object to which to assign the return value that indicates the point in the list where the subform resides. This parameter is 1-based. Set the parameter to an applicable object from the object list.

Returns

The system function returns the index value of the subform to the object defined by Index.

Set Selected Wizard Page

Use this system function to make one of the wizard pages active; that is, "selected."

Parameters

Wizard Control

The wizard FC to affect.

Subform

Input, required. The page you want to make active. Set the parameter to an applicable object from the object list.

Additional Notes

This function is specifically designed to enable the user to change the initial page that will be displayed on a wizard control. If the Enable Re-entry Save option is selected, you can use this system function to display the page where the user left off when he or she restarts the wizard, for example.

Set Selected Wizard Page should not be called outside of the initialization of a wizard control (**Wizard is Initialized** or **Wizard Post-Initialized** events). Calling this function outside of the initialization of a wizard control might cause the wizard to malfunction and so is unsupported.

If a page is visible and enabled, this function enables you to skip it only if it has a status of complete or incomplete.

Set Wizard Form Mode

Use this system function to change the runtime mode of the wizard form to update, add, or copy. Use this function in the **Wizard is Initialized** event only.

Parameter

Mode

Input, required. The runtime mode to which to set the wizard form. Set the parameter to <Update Mode> (1), <Add Mode> (2), or <Copy Mode> (3).

Set Wizard Page Index

Use this system function to change the index of a selected wizard page, effectively "reordering" the pages. The list of wizard pages starts with an index of one.

Parameters

Wizard Control

The wizard FC to affect.

Subform

Input, required. The page for which you want to change the index value. Set the parameter to an applicable object from the object list.

Index

Input (integer), required. The index value to assign to the page. This parameter is 1-based. Set the parameter to an applicable object from the object list.

Set Wizard Page Status

Use this system function to set the status of a page in the wizard. The default status of a wizard page is to have no status.

Parameters

Subform

Input, required. The subform FC to affect.

Status

Input (integer), required. The status to which to set the page. Set the parameter to <Complete> (1), <Incomplete> (2), or applicable object from the object list.

Additional Notes

The default status of a wizard page is to have no status. An enabled and visible page with no status will not permit the user to move the next page. Changing the status to either complete or incomplete will permit the user to move to the next page.

All enabled and visible pages must be set to a complete status in order for completion processing to take place and permit the user to complete the wizard.

Suppress Wizard Page Validation and Save

Use this system function to prevent runtime validation and save functions for a specified wizard page. Typically, you suppress validation and save on invisible or disabled wizard pages.

Parameters

Subform

Input, required. The subform FC to affect.

Suppress

Input (integer), required. A flag indicating whether to validate and save the page. Set the parameter to **<Yes>** (1) or **<No>** (2).

A

System Functions in Form Design Aid

This appendix contains the following topic:

- [Section A.1, "System Functions"](#)

This appendix discusses most of the system functions that you can access through Form Design Aid (FDA).

A.1 System Functions

You access and apply most system functions through Form Design Aid (FDA). System functions are visually grouped into folders based on function or control type. This appendix mirrors that system function folder organization; however, it does not describe the system functions that are detailed in a chapter devoted to a particular control type. Furthermore, some system functions are described in other locations entirely.

Note: Do not use the system functions in the Deprecated or Obsolete folders.

Control

These system functions are applied on the control level and work for most control types. System functions that apply to a specific control type do not appear here; they are described in the chapter dedicated to that control type.[Edit Control System Functions](#)
[Tab Control System Functions](#)

Clear Control Error

Use this system function to clear the errors that have been set on a control.

Parameter

Control

Input, required. The form control (FC) to affect.[Set Control Error](#)

Disable Control

Use this system function to render a control unavailable for entry both by the user and programmatically. A disabled control is still visible.

Parameter

Control

Input, required. The FC to affect.[Enable Control](#)

Enable Control

Use this system function to render a control available for entry both to the end user and programmatically.

Parameter

Control

Input, required. The FC to affect.[Disable Control](#)

Go to Url

Use this system function to insert a functional URL into the application.

Parameter

URL

Input, required. The fully-qualified URL to which to link. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Hide and Reclaim Space

Use this system function to hide a subform or groupbox and reclaim the space.

Parameters

Control

Input, required. The FC (subform or groupbox) to affect.

Additional Notes

The function will reclaim space by adjusting controls below the Input FC. The space for the Input FC will not be reclaimed if there are adjacent controls.

Hide Control

Use this system function to prevent the user from seeing (and therefore interacting with) a control. Hidden controls can be manipulated programmatically.

Parameter

Control

Input, required. The FC to affect.[Disable Control](#)[Show Control](#)

Set Control Error

Use this system function to set an error on a control.

Parameters

Control

Input, required. The FC to affect.

Error Code

Input, required. The error to set on the control. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.[Clear Control Error](#)

Set Control Text

Use this system function to change the label of a control in a given instance.

Parameters

Control

Input, required. The FC to affect.

Text

Input, required. The text to show as the label for the control. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.[Clear Control Error](#)

Set Data Dictionary Item

Use this system function to select the data dictionary (DD) item to use for a control.

Parameters

Control

Input, required. The FC to affect.

DD Alias

Input, required. The DD item to which to apply to the control. Set the parameter to select a DD item from the Data Dictionary dialog (<Pick DD Item>) or an applicable object from the object list.

System Code

Input, required. The system code to use when determining whether to apply system code-based jargon to the DD item text fields. Set the parameter to match the current system code (<Default>) or an applicable object from the object list.

Additional Notes

The control must be a DD item, not database item. When changing them, the DD items to be switched must be of the same type except for one case: You can change a string to a character but not vice versa. If you make a change between string types, the maximum size of the new string item will be the smaller size of the two switched items.

Set Data Dictionary Overrides

Use this system function to apply DD overrides of any type to a control based on any kind of data item (included BV items).

Parameters

Control

Input, required. The FC to affect.

Overrides

Input, required. The override to apply. Set the value to <Data Dictionary Overrides>, or double-click <Data Dictionary Overrides> to select specific overrides to set.[Set Control Text](#)

Set Statusbar Text

Use this system function to set the text in the status bar for a given control, identified by its associated DD item.

Parameter

DD Alias

Input, required. The DD item to affect. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, or <Zero>.

Additional Notes

To clear the status bar text, use this system function with a DD Alias parameter of <Blank> or <Zero>.

Show Control

Use this system function to enable the user to see (and therefore interact with) a hidden control.

Parameter

Control

Input, required. The FC to affect.[Set Control Text](#)

Was Value Entered

Use this system function to determine if a form control has been changed. A return value of zero indicates no change, and a return value of one indicates a change.

Parameters

Control

Input, required. The FC to affect.

Return To

Input, required. The object to which to return the Boolean. Set the parameter to an applicable object from the object list.

Additional Notes

Two flags track the changes: the form flag and the control flag. When a control is changed, both flags are set to one. When **Was Value Entered** is called and a specific control is selected for the control parameter, the current value of the control flag is returned. The control value is then set to zero. The form flag remains the same. When **<All Controls>** is selected for the control parameter, the current value of the form flag is returned. The form flag value is then set to zero, but the control flag remains the same.

Returns

This system function can return one of these values:

- 0
The control did not change.
- 1
The control changed.

Expand Group Box

Use this system function to expand a group box.

Parameter

GroupBox

Input, required. The FC to affect.

Note: The collapsible functionality needs to be activated in the group box properties to use this system function. To expand all expandable group boxes at once select <All Collapsible GroupBoxes> from Available Objects.

Collapse Group Box

Use this system function to collapse a group box.

Parameter

GroupBox

Input, required. The FC to affect.

Note: The collapsible functionality needs to be activated in the group box properties to use this system function. To collapse all expandable group boxes at once select <All Collapsible GroupBoxes> from Available Objects.

General

These system functions affect applications in a variety of ways. They reside in the General folder in FDA.

Cancel User Transaction

Use this system function to cancel any transaction committed by the user. The current transaction is assumed to be the one to cancel, so no input parameters are necessary.

Continue Custom Data Fetch

Use this system function to set up a custom fetch routine for page-at-a-time processing. Called on the **Get Custom Grid Row** event, it causes runtime to add lines to the grid (one at a time) until the page is full. No input parameters are necessary.

Copy Currency Information

Use this system function to copy currency information (type and number of decimal places) between controls.

Parameters

To Control

Input (math numeric), required. The control to which to copy the currency data. Set the parameter to an applicable object from the object list.

From Control

Input (math numeric), required. The control from which to copy the currency data. Set the parameter to an applicable object from the object list.

Dynamic Form Interconnect (Web Only)

Use this system function to call another form when the application ID, form ID, and data structure values are not known until runtime.

Parameters

Application ID

Input (string), required. The ID of the calling application.

Form ID

Input (string), required. The identifier of the called form data structure.

Version

Input (string), optional. The version of the called form.

DS Value

Input (string), optional. Contains all the data structure values needed to be passed to the form in the format id | value | id | value... and so on.

Get VCard (Release 9.2.1)

Use this system function within a hover form to get the VCard information, including the VCard string and VCard name, that was set on the static text control using the Set VCard system function.

Parameters

Static Text

Input (string), required. The name of the static text control.

VCard String

Output (string). The content of the VCard that is returned.

VCard Name

Output (string). The name of the VCard that is returned.

Launch Batch Application

Use this system function to establish a report interconnection and launch a batch application (report).

Parameters

Report Name

Input (string), required. The report to launch. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Version Name

Input (string), required. The version of the report to launch. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Print Preview?

Input (character: Y/N), required. An indicator of whether to display a print preview of the report. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Data Selection?

Input (character: Y/N), required. An indicator of whether to provide the user the opportunity to override the default data selection for the report. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Data Sequencing?

Input (character: Y/N), required. An indicator of whether to provide the user the opportunity to override the default data sequencing for the report. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Push Specs Only?

Input (character: Y/N), required. An indicator of whether to submit the specifications from the version only. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

PO Template Name

Input (string), required. The processing option data structure to use. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Prompt for Values?

Input (character: Y/N), required. An indicator of whether to prompt the user for processing option values. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Date Last Executed

Input (JDEDATE), required. The source for the date indicating when the report was run. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Data Source Override

Input (string), required. The source for the data upon which to base the report. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

JDE Log?

Input (character: Y/N), required. An indicator of whether to generate a JDE.log file. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

JDEDbug Log?

Input (character: Y/N), required. An indicator of whether to generate a Jdedebug.log file. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

UBE Logging Level?

Input (integer), required. The level of detail to apply when compiling Jdedebug.log. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Jargon Code

Input (string), required. The system code corresponding to the jargon values you want to apply. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Cover Page?

Input (character: Y/N), required. An indicator of whether to print a cover page for the report. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Job Queue Name

Input (string), required. The name to use to identify the job after it has been submitted to the job queue. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

TC Prompting?

Input (character: Y/N), required. An indicator of whether to prompt the user for table conversion option values. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Process Type

Input (character), required. The type of process to run. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Launch Processing Options Dialog

When launching a batch application (report), use this system function to launch the processing options dialog for it.

Parameters

Object Name

Input, required. The report being launched. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Version Name

Input, required. The version of the report being launched. Set the parameter to an alphanumeric constant (<Literal>), <Null>, or an applicable object from the object list.

Press Button

Use this system function to "click" a button in the current application programmatically.

Parameter

Control

Input, required. The button to "click." Set the parameter to an applicable object from the object list.

Additional Notes

This is applicable for enabled (but not necessarily visible) form controls and HCs. The event, Button Clicked, fires for the control that is pushed. Note that this system function moves the focus to the control that was activated.

Run Executable

Use this system function to launch an executable program outside of JD Edwards EnterpriseOne.

Parameters

EXE Directory

Input (string), required. The location of the executable. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

EXE Name

Input (string), required. The name of the executable. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Parameters #1

Input (string), required. The first input parameter to pass to the executable. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Parameters #2

Input (string), required. The second input parameter to pass to the executable. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Parameters #3

Input (string), required. The third input parameter to pass to the executable. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Working Directory

Input (string), required. The location where the executable should place its temporary files. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Send Email (Release 9.2.1)

Use this system function within a hover form to send an email to the list of recipients.

Parameters

Recipients

Input (string), required. A list of recipients to send the email to.

Show Parameterized URL

Input, required. Value is <YES> or <NO>. If <YES>, the parameterized URL is attached to the email.

Send Meeting Request (Release 9.2.1)

Use this system function within a hover form to send a meeting request to the list of recipients.

Parameters

Recipients

Input (string), required. A list of recipients to send the meeting request to.

Show Parameterized URL

Input, required. Value is <YES> or <NO>. If <YES>, the parameterized URL is attached to the meeting request.

Set Control Focus

Use this system function to place focus on a specific control.

Parameter

Control

Input, required. The control to receive the focus. Set the parameter to an applicable object from the object list.

Additional Notes

This is applicable for form controls and HCs. The event, **Control Is Entered**, fires on the control that receives focus.

Set Form Title

Use this system function to change the title of the current form.

Parameter

New Title

Input, required. The title to display on the form. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Dynamic Form Interconnect (Web Only)

Use this system function to call another form when the application ID, form ID, and data structure values are not known until runtime.

Parameters

Application ID

Input (string), required. The ID of the calling application.

Form ID

Input (string), required. The identifier of the called form data structure.

Version

Input (string), optional. The version of the called form.

DS Value

Input (string), optional. Contains all the data structure values needed to be passed to the form in the format id | value | id | value... and so on.

Set Modified Web Object Behavior

Use this system function from a One View Financial Statements application to interact with Queries and One View Reporting, all of their related operations, and to export or import One View Financial Statements data.

Note: With the exception of the import and export functionality, this system function can be used by any One View Reporting application, not just One View Financial Statements applications.

The parameters for this system function vary depending on the Action parameter. The Action parameter defines how the system function is being used.

The various actions for this system function can be organized by web object type:

- QUERY as Web Object Type
 - LOAD_NAMED_AQ
 - DOES_AQ_EXIST
 - PREVIEW (for Web Object Type QUERY)
 - DELETE_AQ_NAME
- ONEVIEW as Web Object Type
 - SUPPRESS_ONEVIEW_MENU
 - LAUCH_MODIFIED_OVR_SIDEVIDEO
 - DOES_OVR_EXIST
 - PREVIEW (for Web Object Type ONEVIEW)
 - DELETE_OVR_NAME
 - FETCH_LAYOUTS
 - FETCH_FORMATS
 - RUN_MODIFIED_OVR
- NONE as Web Object Type (Export/Import)
 - EXPORT_FRW_RPT
 - EXPORT_FRW_ROW
 - EXPORT_FRW_COL
 - IMPORT_FRW_RPT
 - IMPORT_FRW_ROW
 - IMPORT_FRW_COL

This is the order in which the various actions and their parameters are discussed in the next sections.

All input and output parameters not used during a specific action require a variable to be passed. The variable should be set to blank or set to the proper input or output value prior to the system function call. Using <BLANK> as an input or output

parameter can cause issues with subsequent calls because the residual value of the parameter remains.

Important: Do not use <BLANK> in the input or output parameters.

All parameters require a variable. Do not pass <BLANK> in any parameter. If a parameter is not listed for the action in the following sections, you should still pass a variable set to blank for that parameter.

Parameters for LOAD_NAMED_AQ

Use to load the modified Query side panel for storing modified metadata. This stops normal Query functionality in the application. The target event is Dialog Initialized or Post Dialog Initialized.

Action

Input, required. Value is LOAD_NAMED_AQ.

Web Object Type

Input, required. Value is QUERY.

Web Object Name

Input. Pass blank as a variable for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

Web Object Name Short

Input, required. This field should be the name of the query.

Application Name

Input, required. Must match calling application.

Form Name

Input, required. Must match calling form.

Version

Input, required. Must match calling form's version. Value may be Blank if there is no version.

Parameters for DOES_AQ_EXIST

Use to check if the passed in query name exists in F952471. The target event is Form/Row menu options or push buttons.

Note: AQEXAML or AQCRITERIAXML returns a formatted query string in the table name field. This can be used to send the information, along with the extracted data, to the BI Publisher server for processing.

Action

Input, required. Value is DOES_AQ_EXIST.

Web Object Type

Input. Value is QUERY.

Web Object Name

Input, required. Pass value to use primary key. Pass a variable set to blank for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, optional. Pass value to use with either primary or unique key. For example, AQEXAXML, AQWRTXML, or AQCRITERIAXML.

The default behavior of this action is to fetch the modified query's EXAXML record in F952471. Since XML Type is part of the primary key and the unique key, a specific valid value should be passed for it. If XML Type is not passed when using the DOES_AQ_EXIST action, then the system will use a default value of "AQEXAXML" to try and fetch the modified query's EXAXML record. If a valid EXAXML record exists in F952471, then it will be fetched. If a user wants to use the DOES_AQ_EXIST action to check if a modified query's WRTXML or CRITERIAXML records exist in F952471, then the XML Type value will have to be passed and it should have a value like "AQWRTXML" or "AQCRITERIAXML."

Web Object Name Short

Input, required. Pass for unique key.

Web Object Name Long

Input, optional.

Application Name

Input, required. Pass for unique key.

Form Name

Input, required. Pass for unique key. This must match LOAD_NAMED_AQ that inserts or updates the data.

Version

Input, required. Pass for unique key.

Table Name

Output. AQEXAXML or AQCRITERIAXML returns a formatted query string to be used with BI Publisher sample data.

Future Use 1

Output. F952471.XMWOFU1

Future Use 2

Output. F952471.XMWOFU2

Web Object Output

Output. F952471.XMOMRBLOB

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for PREVIEW When Used for Modified Query Preview

Use to load the query in preview/read only mode. The target event is Dialog Initialized or Post Dialog Initialized.

Action

Input, required. Value is PREVIEW.

Web Object Type

Input, required. Value is QUERY.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

Web Object Name Short

Input, required. This field should be the name of the query.

Application Name

Input, required. Must match calling application.

Form Name

Input, required. Must match calling form.

Version

Input, required. Must match calling form's version. Value can be Blank (as a variable) if there is no form version.

Parameters for DELETE_AQ_NAME

Use to delete a query attached to a One View Financial Statements component. The target event is Form/Row menu options or push buttons.

Action

Input, required. Value is DELETE_AQ_NAME.

Web Object Name

Input, optional. Pass value to use primary key. Pass variable set to blank for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, optional. Pass value to delete a specific record. Otherwise, pass variable set to blank to delete all records.

Web Object Name Short

Input, required. Pass for unique key.

Application Name

Input, required. Pass for unique key.

Form Name

Input, required. Pass for unique key.

Version

Input, required. Pass for unique key.

Error Message

Output. Returns blank when successful or an error code when it fails.

Parameters for SUPPRESS_ONEVIEW_MENU

Use to suppress the standard One View menu. SUPPRESS_ONEVIEW_MENU is a prerequisite to calling the LAUNCH_MODIFIED_OVR_SIDEVIDEO and RUN_MODIFIED_OVR system functions. The target event is Dialog Initialized or Post Dialog Initialized.

Action

Input, required. Value is SUPPRESS_ONEVIEW_MENU.

Parameters for LAUNCH_MODIFIED_OVR_SIDEVIDEO

Use to launch the modified One View Reporting side panel for One View Financial Statements. The target event is Dialog Initialized or Post Dialog Initialized.

Action

Input, required. Value is LAUNCH_MODIFIED_OVR_SIDEVIDEO.

Web Object Type

Input, required. Value is ONEVIEW.

Web Object Name

Input, required. Pass a value to use the primary key. Pass variable set to blank for a unique key.

Web Object User

Input, required. If not passed, defaults to *PUBLIC.

XML Type

Input, required. Value is OVRDESIGNXML.

XML Type is a mandatory field that has to be passed both when using Primary Key or Unique Key. If it is not passed when invoking the LAUNCH_MODIFIED_OVR_SIDEVIDEO, then the system will, by default, use the value of "OVRDESIGNXML". A literal String "OVRDESIGNXML" can be used as the default because for a particular report definition there is going to be only one "OVRDESIGNXML" record in F952471 at any point of time.

Web Object Name Short

Input, optional. Pass for unique key or pass variable set to blank.

Application Name

Input, optional. Pass for unique key or pass variable set to blank.

Form Name

Input, optional. Pass for unique key or pass variable set to blank.

Version

Input, optional. Pass for unique key or pass variable set to blank.

Future Use 1

Output. Pass blank as a variable or pass 1.

In F952471 table, the Future Use Flag 1 (WOFU1) can either have a blank value or it can have a value of "1". If it has no value/ blank value, then it means that a BIP Layout has not yet been created for that FRW Report. When this field has no value the Modified OVR side panel launches in Create Mode. When a BI Publisher layout is created, the system updates this field to a value of "1". The value of "1" is used to launch the Modified OVR side panel in Edit Mode.

Future Use 2

Output. Pass blank as a variable or pass 1.

In the F952471 table, the Future Use Flag 2 (WOFU2) has no value (blank value) to start with. Once a BI Publisher report has been created for a One View Financial Statements report, if the underlying xml for the report changes, then the system updates this field to a value of "1". When a user launches the Modified OVR side panel and clicks on the Edit icon in the side panel to launch the BI Publisher Layout Editor, the system checks for the value of this field before launching the BI Publisher Layout Editor. If the value of this field is "1", then the "sample.xml" is regenerated using the new xml and uploaded to BI Publisher for the already created report so that the layout can be designed using the updated xml. Once the new "sample.xml" is generated and uploaded to BI Publisher, the system resets the value of this field to "0", so that the "sample.xml" will not be regenerated again when a user clicks on the Edit icon next time unless another change in XML is made. This field is updated to "1" whenever there is a change in XML so that the "sample.xml" can be regenerated using this new XML.

Parameters for DOES_OVR_EXIST

Use to check if the passed in One View report name exists in F952471. The target event is Form/Row menu options or push buttons.

Action

Input, required. Value is DOES_OVR_EXIST.

Web Object Type

Input, optional. Value is ONEVIEW.

Web Object Name

Input, required. Pass value to use primary key. Pass variable set to blank for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, optional. Pass value to use with either primary or unique key. For example, OVRDESIGNXML.

The default behavior of this action is to fetch the modified One View design-time xml record in F952471. Since XML Type is part of the primary key and the unique key, a specific valid value should be passed for it. If XML Type is not passed when using the DOES_OVR_EXIST action, then the system will use a default value of "OVRDESIGNXML" to try and fetch the modified One View design-time xml record. If a valid design-time xml record exists in F95247, then it will be fetched. If a user wants to use the DOES_OVR_EXIST action to check if a modified One View runtime xml record exists in F952471, then the XML Type value will have to be passed and it should have a value like "OVRRUNXML+RUNID" where "OVRRUNXML" is a static String and "RUNID" is dynamically generated. Since "RUNID" is dynamically generated by the Applications Development teams and there can be more than one runtime xml record for the same report definition, the system cannot use a default value for XML Type.

Web Object Name Short

Input, required. Pass for unique key or pass variable set to blank.

Application Name

Input, required. Pass for unique key.

Form Name

Input, required. Pass for unique key.

Version

Input, required. Pass for unique key.

Future Use 1

Output. F952471.XMWOFU1

Future Use 2

Output. F952471.XMWOFU2

Web Object Output

Output. F952471.XMOMRBLOB

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for PREVIEW When Used for Modified One View Reporting Preview

Use to load the modified One View Report in preview/read only mode. The target event is Dialog Initialized, Post Dialog Initialized, or Button Click events.

Action

Input, required. Value is PREVIEW.

Web Object Type

Input, required. Value is ONEVIEW.

Web Object Name

Input, required. Pass a value to use the primary key. Pass variable set to blank for a unique key.

Web Object User

Input, required. If not passed, defaults to *PUBLIC.

XML Type

Input, required. Value is OVRDESIGNXML.

Web Object Name Short

Input, optional. Pass for unique key or pass variable set to blank.

Application Name

Input, optional. Pass for unique key or pass variable set to blank.

Form Name

Input, optional. Pass for unique key or pass variable set to blank.

Version

Input, optional. Pass for unique key or pass variable set to blank.

Parameters for DELETE_OVR_NAME

Use to delete a One View report attached to a One View Financial Statements component. The target event is Form/Row menu options or push buttons.

Note: DELETE_OVR_NAME deletes the report template and data model on the BI Publisher server. The OVRDESIGNXML record is required for deleting BI Publisher artifacts. Do not manually delete the F952471 because this results in orphaned BI Publisher artifacts.

Action

Input, required. Value is DELETE_OVR_NAME.

Web Object Name

Input, optional. Pass value to use primary key. Pass variable set to blank for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, optional. Pass value to delete a specific record. Otherwise, pass blank as a variable to delete all records.

Web Object Name Short

Input, required. Pass for unique key.

Application Name

Input, required. Pass for unique key.

Form Name

Input, required. Pass for unique key.

Version

Input, required. Pass for unique key.

Error Message

Output. Returns blank when successful or an error code when it fails.

Parameters for FETCH_LAYOUTS

Use to fetch the layout names from the BI Publisher server. The target event is Dialog Initialized or Post Dialog Initialized.

Action

Input, required. Value is FETCH_LAYOUTS.

Web Object Name

Input, required. Pass value to use primary key. Pass blank as a variable for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, required. Value is OVRDESIGNXML.

Web Object Name Short

Input, optional. Pass for unique key or pass variable set to blank.

Application Name

Input, optional. Pass for unique key or pass variable set to blank.

Form Name

Input, optional. Pass for unique key or pass variable set to blank.

Version

Input, optional. Pass for unique key or pass variable set to blank.

Future Use 1

Output. Value is default layout.

Future Use 2

Output. Value is default format.

Web Object Output

Output. Value is a list of layouts delimited by pipe "|".

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for **FETCH_FORMATS**

Use to fetch the formats from the BI Publisher server for a specific layout that was retrieved by **FETCH_LAYOUTS**. The target event is Tab out of control (for the selected layout).

Action

Input, required. Value is **FETCH_FORMATS**.

Web Object Name

Input, required. Pass value to use primary key. Pass blank as a variable for unique key.

Web Object User

Input, optional. If not passed, defaults to *PUBLIC.

XML Type

Input, required. Value is OVRDESIGNXML.

Web Object Name Short

Input, optional. Pass for unique key or pass variable set to blank.

Application Name

Input, optional. Pass for unique key or pass variable set to blank.

Form Name

Input, optional. Pass for unique key or pass variable set to blank.

Version

Input, optional. Pass for unique key or pass variable set to blank.

Future Use 1

Output. Must be an existing layout from **FETCH_LAYOUTS**.

Web Object Output

Output. Value is a list of formats delimited by pipe "|".

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for RUN_MODIFIED_OVR

Use to run the One View Financial Statements, which will in turn kick off the BI Publisher report. The target event is Button Click or Form Exit Click.

Note: SUPPRESS_ONEVIEW_MENU action must be called as a prerequisite for this run action in the Dialog is Initialized or Post Dialog is Initialized event.

The application name should be the same as the application that was used to create report in BI Publisher. In order to fetch the report in BI Publisher, the system uses a report URL (for example, /JD Edwards/DV910/P09330/Report1.xdo so the application name should always be the same at both report design and runtime).

The layout and format with which the report must be run, have to be passed in the Future Use 1 and 2 fields respectively. If they are not passed, the default layout and default format are used to run the report.

Action

Input, required. Value is RUN_MODIFIED_OVR.

Web Object Type

Input, required. Value is ONEVIEW.

Web Object Name

Input, required. Pass value to use primary key. Pass blank as a variable for unique key.

Web Object User

Input, required. If not passed, defaults to *PUBLIC.

XML Type

Input, required. Value is OVRDESIGNXML.

XML Type is a mandatory field to be passed both when using Primary Key or Unique Key for the RUN_MODIFIED_OVR action. It should be passed and it should have a value like "OVRRUNXML+RUNID" where "OVRRUNXML" is a static String and "RUNID" is dynamically generated. The length of XML Type is 20 characters. Please note that RUNID should not exceed 11 characters. Since "RUNID" is dynamically generated by the Applications Development teams and there can be more than one runtime xml record for the same report definition, the system cannot use a default value for XML Type for the RUN_MODIFIED_OVR action.

Web Object Name Short

Input, optional. Pass for unique key or pass variable set to blank.

Application Name

Input, optional. Pass for unique key or pass variable set to blank.

Form Name

Input, optional. Pass for unique key or pass variable set to blank.

Version

Input, optional. Pass for unique key or pass variable set to blank.

Future Use 1

Input, required. Must be an existing layout in BI Publisher for passed report and it must be present in list of layouts from FETCH_LAYOUTS.

Future Use 2

Input, required. Must be one of the supported One View Financial Statement formats. For example, analyze, pdf, rtf, xlsx, or pptx.

Parameters for EXPORT_FRW_RPT

Use to export the F952471 advanced metadata, BI Publisher report template, and data model; F09330 report definition; F09310 report row definitions; F09320 report column definitions; and F09340 report version data in a zip format. Once the zip file is created, the file is sent to the browser for download. The zip file allows transport or transfer of information to another environment via import in the specified environment. The target event is Row Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is EXPORT_FRW_RPT.

Web Object Name

Input, not necessary if Web Object Short Name is present. Then pass variable set to blank.

Web Object User

Input, not necessary. If not passed, defaults to *PUBLIC.

Web Object Name Short

Input, not necessary if Web Object Name is present. Then pass variable set to blank.

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for EXPORT_FRW_ROW

Use to export the F952471 advanced metadata and F09310 row definition data in a zip format. Once the zip file is created, the file is sent to the browser for download. The zip file allows transport or transfer of information to another environment via import in the specified environment. The target event is Row Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is EXPORT_FRW_ROW.

Web Object Name Short

Input, Row Definition Name.; if exporting multiple, use "|" delimiter with string of multiple Row Definition Names.

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for EXPORT_FRW_COL

Use to export the F952471 advanced metadata and F09320 column definition data in a zip format. Once the zip file is created, the file is sent to the browser for download. The zip file allows transport or transfer of information to another environment via import

in the specified environment. The target event is Row Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is EXPORT_FRW_COL.

Web Object Name Short

Input, Column Definition Name.; if exporting multiple, use " | " delimiter with string of multiple Row Definition Names.

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for IMPORT_FRW_RPT

Use to import the F952471 advanced metadata, BI Publisher report template, and data model; F09330 report definition; F09310 report row definitions; F09320 report column definitions; and F09340 report version data from the .zip file into the database tables. The end user will be required to choose the file for import. A message box will appear requesting user interaction if there are record conflicts. The target event is Form Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is IMPORT_FRW_RPT.

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for IMPORT_FRW_ROW

Use to import the F952471 advanced metadata and F09310 row definition data from the .zip file into the database tables. The end user will be required to choose the file for import. A message box will appear requesting user interaction if there are record conflicts. The target event is Form Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is IMPORT_FRW_ROW.

Error Code

Output. Returns blank when successful or an error code when it fails.

Parameters for IMPORT_FRW_COL

Use to import the F952471 advanced metadata and F09320 column definition data from the .zip file into the database tables. The end user will be required to choose the file for import. A message box will appear requesting user interaction if there are record conflicts. The target event is Form Menu, Push Button, or press Grid Icon.

Action

Input, required. Value is IMPORT_FRW_COL.

Error Code

Output. Returns blank when successful or an error code when it fails.

Set Time Zone On Form

Use this system function to set the time zone for the current form.

Parameter

Time Zone

Input, required. The time zone to use for the current form. Set the parameter to an applicable object from the object list.

Set VCard (Release 9.2.1)

Use this system function within a hover form to set the VCard information for a static text control.

Parameters

Static Text

Input (string), required. The name of the static text control.

VCard String

Input (string), required. The content of the VCard.

VCard Name

Input (string), required. The name of the VCard.

Stop Processing

Use this system function to stop runtime from processing the ER on the current event.
No parameters are necessary.

Suppress Add

Use this system function to prevent the runtime engine from executing a database add. No parameters are required. Call this system function on the **Add Rec to DB - Before** or **Add Grid Rec to DB - Before** event rule, as appropriate.

Suppress Default Visual Assist Form

Use this system function to prevent the default form from appearing when the user clicks a visual assist. No parameters are required. Call this system function on the **Visual Assist Button Clicked** event, followed by a call to the form that you want to open instead.

Suppress Delete

Use this system function to prevent the runtime engine from executing a database delete. No parameters are required. Call this system function on the **Delete Rec to DB - Before** or **Delete Grid Rec to DB - Before** event rule, as appropriate.

Suppress Find

Use this system function to prevent the runtime engine from executing a database fetch. No parameters are required.

Suppress Update

Use this system function to prevent the runtime engine from executing a database update. No parameters are required. Call this system function on the **Update Rec to DB - Before** or **Update Grid Rec to DB - Before** event rule, as appropriate.

Time Between

Use this system function to calculate the amount of time that passed between two dates.

Parameters

Start UTC

Input (JDEUTime), required. The first date in Universal Time Code (UTC). Set the parameter to an alphanumeric constant (**<Literal>**) or an applicable object from the object list.

End UTC

Input (JDEUTime), required. The second date in Universal Time Code (UTC). Set the parameter to an alphanumeric constant (**<Literal>**) or an applicable object from the object list.

Days

Input, required. The object to which to assign the number of days that have passed. Set the parameter to an applicable object from the object list.

Hours

Input, required. The object to which to assign the number of hours that have passed. Set the parameter to an applicable object from the object list.

Minutes

Input, required. The object to which to assign the number of minutes that have passed. Set the parameter to an applicable object from the object list.

Seconds

Input, required. The object to which to assign the number of seconds that have passed. Set the parameter to an applicable object from the object list.

Returns

This system function returns the difference in days, hours, minutes, and seconds between two dates to the objects identified by Days, Hours, Minutes, and Seconds, respectively.

Was Form Record Fetched

Use this system function to determine whether the interactive engine for the current form fetched a form record.

Parameter

Return To

Input, required. The object to which to return the value. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- 0
Either no fetch attempt was made, or a fetch attempt failed.
- 1
A record was fetched successfully.

Messaging

You should use only one of the system functions in this group: **Send Message Extended**. The other system functions in the Messaging folder are intended for internal JD Edwards development use only.

Send Message Extended

This system function enables your application to send email messages to users, groups, and so forth.

Parameters

To recipient

Input, optional. The account or accounts to which to send the email.

Cc recipient

Input, optional. The account or accounts to which to send a courtesy copy of the email.

Bcc recipient

Input, optional. The account or accounts to which to send a blind courtesy copy of the email.

Mailbox

Input, required. The mailbox name to which to deliver the email. The mailbox is used only if the mail is delivered to Work Center. For mail delivered externally (such as SMTP mail), this parameter is ignored. Set the parameter to a specific mailbox or to an applicable object from the object list.

Subject

Input, required. The text to display in the subject line of the email. If the value is <Blank> or <Zero>, and you are basing the email on a DD item using the Message parameter, then the system sets the subject line to the DD item description, if one exists.

Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Text

Input, required. The text to display in the body of the email. Set the parameter to an alphanumeric constant (<Literal>), <Blank>, <Zero>, or an applicable object from the object list.

Shortcut

Input, required. A link to a JD Edwards EnterpriseOne application. Set the parameter to the application to which to link or to <None>.

Message

Input, optional. The text to display in the body of the email, based on a DD item glossary. The recipient formatting preferences (for dates, times, and numeric values) as well as language preference (should a translation for this DD item be available) are used when composing the text that represents the message. Set the parameter to the DD item you want to use, or to <None>.

Media Object Name

Input, optional. The name of the media object to include in the email. Set the parameter to an applicable object from the object list, or to <None>.

Media Object Key

Input, optional. The key of the media object to include in the email. Set the parameter to an applicable object from the object list, or to <None>.

Additional Notes

The **Send Message Extended** system function supports multiple ways to define the recipient of a mail. You can dictate that the message is for a limited group (such as individuals, distribution lists, and so forth), or you can make the recipients dynamic. The delivery method is based on each user's email preferences. You must send the message using at least one of the recipient parameters, although which one you use is immaterial to the system.

When mapping a recipient parameter, these options are available:

- **AB Number**

To send a message to a single user, enter the address book number of a user as the recipient. The mail will be sent to the default contact (contact number 0) for this address book number. Set the parameter to an applicable object from the object list.

Note: JD Edwards EnterpriseOne version 8.10 applications do not employ contacts; therefore, email is sent directly to a user based on the address book number.

- **Contact**

To send a message to an individual in a user's contact list, enter the address book number of a user and then the number of the contact. Set the parameters to an applicable object from the object list.

Note: JD Edwards EnterpriseOne version 8.10 applications do not employ contacts; therefore, this parameter has no effect.

- **Grouped Distribution List**

To send a message to the members of a distribution list, enter the address book number of the list and its structure type. Set the parameters to an applicable object from the object list.

- **Hierarchical Distribution List**

To send a message to the members of a hierarchical distribution list, enter the distribution list structure type, and the address book number of the node to start from in the list. Set the parameters to an applicable object from the object list.

Note: This option is available only from within the JD Edwards EnterpriseOne workflow modeler.

- **SMTP Address**

To send a message to a single user, enter the SMTP address of the user as the recipient. Set the parameter to an applicable object from the object list.

- **Define Dynamic Recipient**

This option enables the selection of any kind of recipient at runtime, as opposed to choosing the kind of recipient at design time (AB Number, Contact, Grouped Distribution List, Hierarchy Distribution List, or SMTP Address).

All the parameters must be mapped to objects from the available object list. At runtime, the recipient is chosen dynamically based on the value of the Recipient Type:

- Recipient Type is '**00**':
This is the equivalent of selecting <**None**>.
- Recipient Type is '**01**':
This is the equivalent of selecting <**Contact**>. The subfields Address book Number and Contact Number are used to determine the recipient.
- Recipient Type is '**02**':
This is the equivalent of selecting <**AB Number**>. The sub field Address book Number is used to determine the recipient.
- Recipient Type is '**03**':
This is the equivalent of selecting <**Grouped Distribution List**>. The subfields Address book Number and Structure Type are used to determine the recipients where Address book Number is the AB number for the distribution list and Structure Type is the organizational structure, based on UDC 01/TS.
- Recipient Type is '**04**':
This is the equivalent of selecting <**Hierarchical Distribution List**>. The subfields Address book Number and Structure Type are used to determine the recipients where Structure Type is the structure and list to use and Address book Number is the point in the hierarchy from which to start.

Note: This hierarchical resolution is available only when you send the email to the JD Edwards EnterpriseOne work center.

- Recipient Type is '**05**':
This is the equivalent of selecting <**SMTP Address**>. The sub field SMTP Address will be used to determine the recipient.
- Other values
Do not use other values, as they are reserved for future use. The list of supported recipient types is defined by the UDC 98/SM.
- None
To not specify a recipient (use None when a recipient is optional).

The body of the email can be preset text (Text), or can be based on a DD item (<**Message**>). In either case, you can include a media object (<**Media Object Name**> and <**Media Object Key**>) and or a link directly to a JD Edwards EnterpriseOne application (<**Shortcut**>) as well.

Attachments can be sent with the mail, by providing the Media Object Name and Media Object Key parameters.

The system function will retrieve the attachments stored within the Media Object specified, and add the data to the mail sent. Only the Media Object 'RTF Text' and 'URL File' attachment types are supported.

Mail Merge & Doc Gen (Web Only)

These system functions enable you to automate mail merge and document generation tasks. They constitute a part of a larger process, as this outline illustrates:

1. Create an RTF in Microsoft Word to use as the template for a merge.

Use Word's field feature to indicate where to place text at merge time. You can create multiple RTF files and then use the CompositeGeneration business function (BSFN B980043) to create a single template from them.

2. Upload the template.

3. Call the **Get XML Data Model** system function to process the template.

Among other files, the system function creates an XML file to populate with data for the merge.

4. Create a business function to populate the XML file and then run it.

5. Run the merge.

Delete Document

Use this system function to delete a generated document.

Parameters

Document ID

Input, required. The document to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the delete. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Display Document

Use this system function to display a generated document to the user.

Parameters

Document ID

Input, required. The document to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the display. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Download Template

Use this system function to download a mail merge template in RTF format so you can edit it. To enable users to save the RTF template from the browser, the template name must not contain spaces.

Parameters

Template ID

Input, required. The template to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the download. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Download Template for Doc Gen

Use this system function to download a document generation template in RTF format so you can edit it.

Parameters

Template ID

Input, required. The template to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the download. Set the parameter to an applicable object from the object list.

Version

Input, required. The version of the template to download. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Get XML Data Model

Use this system function to process a template after uploading. This system function breaks the template into three parts: an XSL file, an XML file, and one or more image files (if any images were included in the template). Then, use a business function to populate the XML file with the data to merge.

Parameters

Template ID

Input, required. The template to affect. Set the parameter to an applicable object from the object list.

XML Data ID

Input, required. The ID to assign to the XML file. Set the parameter to an applicable object from the object list.

Data Type

Input, required. The template type. Set the parameter to <Mail Merge> or <Doc Gen>.

Status

Output (string), required. The object to which to assign the code that indicates the success of the acquire. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Run Doc Gen and Display

Use this system function to run a document generation operation and display the results to the user. You can choose to save the resulting document.

Parameters

Template ID

Input, required. The document generation template to use. Use an already-downloaded template. Set the parameter to an applicable object from the object list.

Data File ID

Input, required. The document generation template to use. Set the parameter to an applicable object from the object list.

Save

Input, required. An indicator of whether to save the results of the operation to a separate file. Set the parameter to <TRUE> or <FALSE>.

Document ID

Input, required. The ID to assign to the document, should you choose to save it. Set the parameter to an applicable object from the object list.

Status

Output (string), required. The object to which to assign the code that indicates the success of the operation. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED

Indicates the process was unsuccessful.

- SUCCESS

Indicates the process completed normally.[Download Template for Doc GenGet XML Data Model](#)

Run Mail Merge and Display

Use this system function to run a mail merge operation that consists of the first two variable sets in the XML file, and then display the results to the user. The resulting document is saved.

Parameters

Template ID

Input, required. The document generation template to use. Use an already-downloaded template. Set the parameter to an applicable object from the object list.

Data File ID

Input, required. The mail merge template to use. Set the parameter to an applicable object from the object list.

Save

Input, required. An indicator of whether to save the results of the operation to a separate file. Set the parameter to <TRUE> or <FALSE>.

Document ID

Input, required. The ID to assign to the resulting document. Set the parameter to an applicable object from the object list.

Status

Output (string), required. The object to which to assign the code that indicates the success of the operation. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.[Download Template](#)[Get XML Data Model](#)

Run Multiple Mail Merge

Use this system function to run a full mail merge operation.

Parameters

Template ID

Input, required. The document generation template to use. Use an already-downloaded template. Set the parameter to an applicable object from the object list.

Data File ID

Input, required. The mail merge template to use. Set the parameter to an applicable object from the object list.

Document ID

Input, required. The ID to assign to the resulting document. Set the parameter to an applicable object from the object list.

Additional Notes

This system function launches an operation that merges all of the data sets in the XML file with the template to create a .pdf. The process runs asynchronously, so it will not overtax the web server. Before running a system function such as **Display Document**, ensure that the operation has completed. To do so, perform a table I/O on column FNDFUF1 in table F980042. If the return string is PENDING, the operation is still running. If the return string is SUCCESS, then the operation is complete.[Download TemplateGet XML Data Model](#)

Upload Template

After designing it, use this system function to upload a mail merge template.

Parameters

Template ID

Input, required. The template to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the upload. Set the parameter to an applicable object from the object list.

File (full path)

Output (string), required. The object to which to assign the full path name of the uploaded file. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Upload Template for Doc Gen

After designing it, use this system function to upload a document generation template.

Parameters

Template ID

Input, required. The template to affect. Set the parameter to an applicable object from the object list.

Return Code

Output (string), required. The object to which to assign the code that indicates the success of the upload. Set the parameter to an applicable object from the object list.

File (full path)

Output (string), required. The object to which to assign the full path name of the uploaded file. Set the parameter to an applicable object from the object list.

Version

Input, required. The version of the template to upload. Set the parameter to an applicable object from the object list.

Returns

This system function can return these values:

- FAILED
Indicates the process was unsuccessful.
- SUCCESS
Indicates the process completed normally.

Glossary

add mode

A condition of a form that enables users to input data.

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.

charts

Tables of information in JD Edwards EnterpriseOne that appear on forms in the software.

edit mode

A condition of a form that enables users to change data.

in-your-face error

In JD Edwards EnterpriseOne, a form-level property which, when enabled, causes the text of application errors to appear on the form.

jde.ini

A JD Edwards EnterpriseOne file (or member for IBM i) 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.

power form

Web-only application forms that enable users to view multiple, interrelated views of data, grids, and tab pages on one form and to pass logic between them.

subform

A subform is a control designed for use on a power form or another subform. Power forms can contain several subforms, so a single power form with multiple subforms enables users to see multiple data views.

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.