



Configuring Siebel Open UI

Siebel Innovation Pack 2013

Version 8.1/8.2, Rev. A

December 2013

ORACLE®

Copyright © 2005, 2013 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, 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 on 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. 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.

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 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.

Contents

Chapter 1: What's New in This Release

Chapter 2: Overview of Siebel Open UI

About Siebel Open UI 17

Differences Between High Interactivity and Siebel Open UI 21

How Siebel CRM Renders High-Interactivity Clients 22

How Siebel CRM Renders Siebel Open UI Clients 24

Summary of Differences Between High Interactivity and Siebel Open UI 28

About Using This Book 30

Important Terms and Concepts 30

How This Book Indicates Computer Code and Variables 31

How This Book Describes Objects 32

About the Siebel Innovation Pack 33

Support for Customizing Siebel Open UI 33

Getting Help from Oracle 35

Chapter 3: Architecture of Siebel Open UI

About the Siebel Open UI Development Architecture 37

Overview of the Siebel Open UI Development Architecture 37

Example of How Siebel Open UI Renders a View or Applet 41

Customizing the Presentation Model and Physical Renderer 43

Stack That Siebel Open UI Uses to Render Objects 46

Items in the Development Architecture You Can Modify 49

Example Client Customizations 50

Differences in the Server Architecture Between High Interactivity and Siebel Open UI 51

Differences in the Client Architecture Between High Interactivity and Siebel Open UI 53

Life Cycle of User Interface Elements 54

Summary of Presentation Model Methods 54

Life Cycle of a Physical Renderer 56

Example of the Life Cycle of a User Interface Element 58

Chapter 4: Example of Customizing Siebel Open UI

Roadmap for Customizing Siebel Open UI 61

Process of Customizing the Presentation Model 62

| | |
|--|----|
| Creating the Presentation Model | 62 |
| Customizing the Setup Logic of the Presentation Model | 64 |
| Customizing the Presentation Model to Identify the Records to Delete | 66 |
| Customizing the Presentation Model to Delete Records | 69 |
| Overriding Predefined Methods in Presentation Models | 74 |
| Customizing the Presentation Model to Handle Notifications | 75 |
| Attaching an Event Handler to a Presentation Model | 78 |
| Customizing Methods in the Presentation Model to Store Field Values | 81 |
| Customizing the Presentation Model to Call the Siebel Server and Delete a Record | 83 |
| Process of Customizing the Physical Renderer | 84 |
| Setting Up the Physical Renderer | 84 |
| Customizing the Physical Renderer to Render List Applets | 86 |
| Customizing the Physical Renderer to Bind Events | 88 |
| Customizing the Physical Renderer to Bind Data | 89 |
| Customizing the Physical Renderer to Refresh the Recycle Bin | 89 |
| Customizing the Event Handlers | 92 |
| Modifying the CSS Files to Support the Physical Renderer | 93 |
| Configuring the Manifest for the Recycle Bin Example | 94 |
| Testing Your Modifications | 95 |

Chapter 5: Customizing Siebel Open UI

| | |
|--|-----|
| Guidelines for Customizing Siebel Open UI | 97 |
| Guidelines for Customizing Presentation Models | 97 |
| Guidelines for Customizing Physical Renderers | 99 |
| Guidelines for Customizing Presentation Models and Physical Renderers | 100 |
| Doing General Customization Tasks | 100 |
| Enabling Object Managers for Siebel Open UI | 101 |
| Preparing Siebel Tools to Customize Siebel Open UI | 104 |
| Modifying the Application Configuration File | 105 |
| Adding Presentation Model Properties That Siebel Servers Send to Clients | 106 |
| Configuring Siebel Open UI to Bind Methods | 110 |
| Calling Methods for Applets and Business Services | 111 |
| Using the Base Physical Renderer Class With Nonapplet Objects | 113 |
| Customizing Events | 118 |
| Creating Components | 119 |
| Allowing Users to Interact with Clients During Business Service Calls | 120 |
| Managing Files | 122 |
| Organizing Files That You Customize | 122 |
| Updating Relative Paths in Files That You Customize | 125 |

Specifying Dependencies Between Presentation Models or Physical Renderers and Other Files 126

Specifying the Order That Siebel Open UI Uses to Download Files 127

Configuring Manifests 128

Overview of Configuring Manifests 128

Configuring Custom Manifests 132

Adding Custom Manifest Expressions 143

Adding JavaScript Files to Manifest Administrative Screens 144

Chapter 6: Customizing Styles, Applets, and Fields

Customizing Client Logo, Background, and Style 145

Customizing the Logo 146

Customizing the Background Image 147

Customizing Browser Tab Labels 150

Using Cascading Style Sheets to Modify Position, Dimension, and Text Attributes of an Object 151

Using Cascading Style Sheet Classes to Modify HTML Elements 153

Customizing the Sequence That Siebel Open UI Uses to Load Cascading Style Sheets 156

Customizing Applets 159

Refreshing Applets That Contain Modified Data 159

Allowing Users to Drag and Drop Data Into List Applets 163

Customizing List Applets to Display a Box List 164

Customizing List Applets to Render as a Carousel 167

Customizing List Applets to Render as a Carousel without Compiling the SRF 173

Customizing List Applets to Render as a Table 176

Configuring the Focus in List Applets 179

Adding Static Drilldowns to Applets 180

Allowing Users to Change the Applet Visualization 182

Displaying Applets Differently According to the Applet Mode 190

Adding Custom User Preferences to Applets 196

Customizing Applets to Capture Signatures 199

Customizing Fields 203

Displaying and Hiding Fields 203

Configuring Spell Checker on Fields 204

Chapter 7: Customizing Calendars and Schedulers

Customizing Calendars 207

Deploying Calendars According to Your Calendar Deployment Requirements 208

Using Fields to Customize Event Styles for the Calendar 208

Customizing Event Styles for the Calendar 211

Customizing Repeating Calendar Events for Siebel Mobile 212

Controlling How Calendars Display Timestamps 213

Replacing Standard Interactivity Calendars 214

Customizing Resource Schedulers 215

Overview of Customizing Resource Schedulers 216

Customizing a Resource Scheduler 217

Customizing the Filter Pane in Resource Schedulers 229

Customizing the Resource Pane in Resource Schedulers 231

Customizing the Timescale Pane in Resource Schedulers 234

Customizing the Schedule Pane in Resource Schedulers 241

Customizing Tooltips in Resource Schedulers 249

Chapter 8: Configuring Siebel Open UI to Interact with Other Applications

Displaying Data from External Applications in Siebel Open UI 253

Displaying Data from External Applications in Siebel Views 253

Displaying Data from External Applications in Siebel Applets 256

Displaying Data from Siebel Open UI in External Applications 260

Displaying Siebel Portlets In External Applications 261

Preparing Standalone Applets 266

Displaying Siebel CRM Applets and Views in External Applications 267

Displaying Siebel CRM Applets in External Applications with Search Criteria 269

Displaying Siebel CRM Applets in Siebel CRM Web Pages 271

Using iFrame Gadgets to Display Siebel CRM Applets in External Applications 272

Chapter 9: Customizing Siebel Open UI for Siebel Mobile

Overview of Customizing Siebel Mobile 275

Mobile Controller and Physical Renderers You Can Modify to Customize Siebel Mobile 275

Third Party JavaScript Plug-Ins You Can Use to Customize Siebel Mobile 276

Templates and Style Sheets You Can Use to Customize Siebel Mobile 277

Setting Up Configuration for Siebel Mobile Examples 277

Determining Whether or Not Siebel Open UI Is Enabled for Siebel Mobile 278

Customizing Layout, Views, Menus, Lists, and Controls 279

Customizing the Layout for Mobile Devices 279

Configuring Views to Use Landscape or Portrait Layout 282

Configuring Siebel Open UI to Display High Interactivity Views in Mobile Web Clients 286

Using Siebel Web Templates to Modify Siebel Mobile Views 288

Customizing Menus and Menu Items 291

Customizing the Number of Columns in Mobile Applets 292

Customizing the Number of Columns in Mobile Tables 295

Customizing Mobile Lists 299

| | |
|---|-----|
| Customizing Tiles | 303 |
| Adding Toggle Controls | 317 |
| Configuring Siebel Open UI to Toggle Row Visibility | 320 |
| Adding the Show More Button to Your Custom Form Applets | 320 |
| Customizing Transitions, Themes, Styles, and Colors | 322 |
| Customizing Transitions That Siebel Open UI Displays When It Changes Views | 323 |
| Customizing Themes That Siebel Open UI Displays in Siebel Mobile Clients | 325 |
| Customizing List Applet Styles | 327 |
| Customizing jQuery Color Swatches That Siebel Open UI Displays in Siebel Mobile Clients | 329 |
| Customizing Scrolling and Swipe | 336 |
| Configuring Generic Scrolling in Siebel Mobile | 337 |
| Configuring Swipe Scrolling | 339 |
| Configuring Infinite Scrolling | 340 |
| Configuring the Height of the Scroll Area | 341 |
| Configuring Swipe to Delete | 342 |
| Customizing How Siebel Open UI Interacts with Siebel Mobile Applications | 343 |
| Adding Maps That Include Location Data in Siebel Mobile | 343 |
| Configuring Siebel Open UI to Display Siebel CRM Data on Google Maps | 346 |

Chapter 10: Customizing Siebel Open UI for Siebel Mobile Disconnected

| | |
|---|-----|
| Overview of Customizing Siebel Open UI for Siebel Mobile Disconnected | 353 |
| Operations You Can Customize When Clients Are Offline | 353 |
| Operations You Cannot Customize When Clients Are Offline | 354 |
| Process of Customizing Siebel Open UI for Siebel Mobile Disconnected | 355 |
| Doing General Customization Tasks for Siebel Mobile Disconnected | 356 |
| Modifying Manifest Files for Siebel Mobile Disconnected | 356 |
| Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence | 359 |
| Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects | 361 |
| Using Custom JavaScript Methods | 364 |
| Using Custom Siebel Business Services | 367 |
| Configuring Data Filters | 369 |
| Configuring Objects That Siebel Open UI Does Not Display in Clients | 369 |
| Configuring Error Messages for Disconnected Clients | 369 |
| Customizing Siebel Pharma for Siebel Mobile Disconnected Clients | 372 |
| Customizing Siebel Service for Siebel Mobile Disconnected Clients | 375 |
| Allowing Users to Commit Part Tracker Records | 376 |

| | |
|---|-----|
| Allowing Users to Return Parts | 378 |
| Allowing Users to Set the Activity Status | 384 |
| Methods You Can Use to Customize Siebel Mobile Disconnected | 387 |
| Methods You Can Use in the Applet Class | 387 |
| Methods You Can Use in the Business Component Class | 390 |
| Methods You Can Use in the Business Object Class | 408 |
| Methods You Can Use in the Business Service Class | 410 |
| Methods You Can Use in the Application Class | 413 |
| Methods You Can Use in the Model Class | 417 |
| Methods You Can Use in the Service Model Class | 418 |
| Methods You Can Use in Offline Classes | 418 |
| Other Methods You Can Use with Siebel Mobile Disconnected | 420 |

Appendix A: Siebel Open UI Application Programming Interface

| | |
|---|-----|
| Overview of the Siebel Open UI Client Application Programming Interface | 423 |
| Methods of the Siebel Open UI Application Programming Interface | 423 |
| Presentation Model Class | 424 |
| Presentation Model Class for Applets | 433 |
| Presentation Model Class for List Applets | 452 |
| Presentation Model Class for Menus | 458 |
| Physical Renderer Class | 460 |
| Business Component Class | 465 |
| Applet Class | 465 |
| Applet Control Class | 466 |
| JQ Grid Renderer Class for Applets | 476 |
| Business Service Class | 477 |
| Application Model Class | 478 |
| Control Builder Class | 489 |
| Locale Object Class | 489 |
| Component Class | 497 |
| Component Manager Class | 500 |
| Cascading Style Sheet Classes | 503 |
| Other Classes | 506 |
| Methods for Pop-Up Objects, Google Maps, and Property Sets | 508 |
| Pop-Up Presentation Models and Physical Renderers | 508 |
| Method That Integrates Google Maps | 514 |
| Methods That Manipulate Property Sets | 518 |

Appendix B: Reference Information for Siebel Open UI

| | |
|---|-----|
| Life Cycle Flows of User Interface Elements | 523 |
| Life Cycle Flows That Save Records | 523 |
| Life Cycle Flows That Handle User Navigation | 525 |
| Life Cycle Flows That Send Notifications | 529 |
| Life Cycle Flows That Create New Records in List Applets | 531 |
| Life Cycle Flows That Handle User Actions in List Applets | 535 |
| Notifications That Siebel Open UI Supports | 541 |
| Summary of Notifications That Siebel Open UI Supports | 542 |
| Using Notifications with Operations That Call Methods | 550 |
| NotifyGeneric Notification Type | 551 |
| NotifyStateChanged Notification Type | 554 |
| Example Usage of Notifications | 557 |
| Property Sets That Siebel Open UI Supports | 562 |
| Siebel CRM Events You Can Use to Customize Siebel Open UI | 563 |
| Internationalization Support | 588 |
| Screens and Views That Siebel Mobile Uses | 590 |
| Screens and Views That Siebel Consumer Goods Uses | 590 |
| Screens and Views That Siebel Sales Uses | 591 |
| Screens and Views That Siebel Service Uses | 593 |
| Screens and Views That Siebel Pharma Uses | 594 |
| Controls That Siebel Open UI Uses | 595 |
| Browser Script Compatibility | 598 |

Glossary

Index

1

What's New in This Release

What's New in Configuring Siebel Open UI, Version 8.1/8.2, Rev. A

Table 1 lists the changes described in this version of the documentation to support this release of the software.

Table 1. What's New in Configuring Siebel Open UI, Version 8.1/8.2, Rev. A

| Topic | Description |
|--|--|
| "Allowing Users to Drag and Drop Data Into List Applets" on page 163 | New topic. Describes how to allow users to drag and drop data in list applets. |
| "Customizing List Applets to Render as a Table" on page 176 | Modified topic. Updated manifest configuration. |
| "Configuring Infinite Scrolling" on page 340 | Modified topic. No additional work is necessary to enable scroll in a single applet. However, you can configure the number of records that Siebel Open UI scrolls. |
| "Controls That Siebel Open UI Uses" on page 595 | New topic. Describes controls that Siebel Open UI uses. |

What's New in Configuring Siebel Open UI, Version 8.1/8.2

Table 2 lists the changes described in this version of the documentation to support this release of the software. The new features described in Table 2 are available in Siebel CRM version 8.1.1.11 and Siebel CRM version 8.2.2.4.

Table 2. What's New in Configuring Siebel Open UI, Version 8.1/8.2

| Topic | Description |
|--|--|
| "Differences Between High Interactivity and Siebel Open UI" on page 21 | New topic. Describes the main differences between clients that use high interactivity and clients that use Siebel Open UI. |
| "Case Sensitivity in Code Examples" on page 32 | New topic. It is recommended that you follow the case-sensitivity rules that standard JavaScript and HTML use. |
| "Life Cycle of a Physical Renderer" on page 56 | Modified topic. Includes a new diagram and information about the methods that a physical render uses. |
| "Preparing Siebel Tools to Customize Siebel Open UI" on page 104 | Modified topic. You must enable the EnableOpenUI parameter so that you can use Siebel Tools to customize Siebel Open UI. |

Table 2. What's New in Configuring Siebel Open UI, Version 8.1/8.2

| Topic | Description |
|--|--|
| "Organizing Files That You Customize" on page 122 | Modified topic. Describes how to use new folders where you can store your customized files. |
| "Updating Relative Paths in Files That You Customize" on page 125 | New topic. If you customize a file, and if you save this custom file in a custom folder, then you must modify any relative paths that this file references. |
| "Allowing Users to Interact with Clients During Business Service Calls" on page 120 | New topic. You can configure Siebel Open UI to make an asynchronous call to a business service. This call allows the user to use the client while the client is waiting for the Siebel Server to reply to this call. |
| "Adding Presentation Model Properties That Siebel Servers Send to Clients" on page 106 | New topic. Describes how to customize presentation model properties. |
| "Configuring Manifests" on page 128 | Modified topic. Siebel Open UI uses a new manifest configuration to identify the files it must download to the client, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4. |
| "Configuring the Focus in List Applets" on page 179 | New topic. If you modify a list applet, then you must make sure that your modification does not adversely affect how Siebel Open UI sets the focus in this list applet. |
| "Adding Static Drilldowns to Applets" on page 180 | New topic. Describes how to add a static drilldown to a list applet. |
| "Allowing Users to Change the Applet Visualization" on page 182 | New topic. Describes how to modify an applet so that the user can change the applet layout. For example, to change the layout from a list to a grid. |
| "Displaying Applets Differently According to the Applet Mode" on page 190 | New topic. Describes how to configure Siebel Open UI to display applets differently according to the applet mode. |
| "Adding Custom User Preferences to Applets" on page 196 | New topic. Describes how to configure Siebel Open UI to customize default applet behavior so that it remembers the actions the user takes that effect this behavior. |
| "Customizing Applets to Capture Signatures" on page 199 | New topic. Describes how to customize applets to capture signatures. |
| "Customizing the Logo" on page 146 | Modified topic. Starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4, Siebel Open UI defines the logo that it displays in the client in CSS files instead of coding the logo in a web template. You can customize this logo. |
| "Customizing the Sequence That Siebel Open UI Uses to Load Cascading Style Sheets" on page 156 | New topic. This configuration allows you to modify the sequence that Siebel Open UI uses to load the cascading style sheets so that it can load them efficiently and logically. |

Table 2. What's New in Configuring Siebel Open UI, Version 8.1/8.2

| Topic | Description |
|---|---|
| “Customizing Browser Tab Labels” on page 150 | New topic. You customize the label that Siebel Open UI displays for the Browser tab. |
| “Replacing Standard Interactivity Calendars” on page 214 | New topic. Describes how to replace some standard- interactivity calendars that do not work properly in Siebel Open UI. |
| “Customizing Resource Schedulers” on page 215 | New topic. Describes how to customize a resource scheduler. |
| “Displaying Data from Siebel Open UI in External Applications” on page 260 | New topic. Describes how to display Siebel CRM applets and views in an external application, including displaying a Siebel portlet. |
| “Third Party JavaScript Plug-Ins You Can Use to Customize Siebel Mobile” on page 276 | Modified topic. Siebel Open UI no longer supports the jQuery Mobile SimpleDialog plug-in, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4. |
| “Customizing the Number of Columns in Mobile Applets” on page 292 | New topic. Describes how to write a custom renderer that modifies the number of maximum columns that Siebel Open UI displays in a Siebel Mobile phone client. |
| “Customizing the Number of Columns in Mobile Tables” on page 295 | New topic. Describes how to use applet user properties or JavaScript to modify the number of columns that Siebel Open UI displays in a Siebel Mobile client. |
| “Customizing Tiles” on page 303 | New topic. Describes how to customize tiles for list applets, mobile lists, and tile scrolling. |
| “Adding the Show More Button to Your Custom Form Applets” on page 320 | New topic. Describes how to configure Siebel Open UI to add the Show More Button to your custom form applets. |
| “Customizing Transitions That Siebel Open UI Displays When It Changes Views” on page 323 | New topic. Describes how to add a custom CSS3 transition. |
| “Customizing Themes That Siebel Open UI Displays in Siebel Mobile Clients” on page 325 | New topic. Describes how to add a custom theme that Siebel Open UI displays on a tablet. |
| “Customizing List Applet Styles” on page 327 | New topic. Describes how to customize list applet styles. |
| “Customizing jQuery Color Swatches That Siebel Open UI Displays in Siebel Mobile Clients” on page 329 | New topic. Describes how to customize the color swatch that Siebel Open UI displays in Siebel Mobile clients. |
| “Configuring Swipe to Delete” on page 342 | Modified topic. Describes how to add a control in Siebel Tools to enable Swipe to Delete. |

Table 2. What's New in Configuring Siebel Open UI, Version 8.1/8.2

| Topic | Description |
|---|---|
| "Customizing Siebel Open UI for Siebel Mobile Disconnected" on page 353 | New chapter. Describes how to customize Siebel Open UI for Siebel Mobile disconnected. |
| "Using the Base Physical Renderer Class With Nonapplet Objects" on page 113 | New topic. The BasePhysicalRenderer class simplifies calls that Siebel Open UI makes to the AttachPMBinding method for nonapplet objects. |
| "Values That You Can Set for the AI Argument" on page 486 | New topic. Describes how to set the ai argument of the InvokeMethod method. |
| Siebel CRM Events You Can Use to Customize Siebel Open UI on page 563 | New topic. Describes reference information for Siebel CRM events that you can use to customize Siebel Open UI. |
| Component Methods | Removed topic. Described methods that are not available for customization. |
| SetProfileAttr Method | Removed topic. Siebel Open UI does not support the SetProfileAttr method. |
| CallServer Method | Removed topic. You can use the InvokeMethod method instead of the CallServer method to call the Siebel Server. For more information, see "InvokeMethod Method for Application Models" on page 485 . |

Additional Changes

This version of Siebel Open UI includes modifications to the data model that differentiate how the manifest handles predefined Siebel CRM records compared to custom records. All topics and examples in this book that described how to use manifest keys have been rewritten to use the new manifest administration screens in the Siebel client. For information about how to migrate your custom manifest configuration from Siebel CRM version 8.1.1.19 or 8.1.1.10 to version 8.1.1.11 or 8.2.2.4, see *Siebel Database Upgrade Guide*.

What's New in Configuring Siebel Open UI, Version 8.1, Rev. A and Version 8.2, Rev. A

Table 3 lists changes described in this version of the documentation to support this release of the software.

Table 3. What's New in Configuring Siebel Open UI, Version 8.1, Rev. A and Version 8.2, Rev. A

| Topic | Description |
|---|--|
| Chapter 4, "Example of Customizing Siebel Open UI" | New chapter. includes a detailed example that describes the typical tasks that you can do to customize Siebel Open UI. |
| "Application Model Class" on page 478 | Revised topic. Includes descriptions of new methods. |
| "Life Cycle Flows of User Interface Elements" on page 523 | New topic. Includes flowcharts you can use to determine the methods that Siebel Open UI uses during various steps in the life cycle of a user interface element. |
| "Notifications That Siebel Open UI Supports" on page 541 | Revised topic. Includes new information about notifications. |

2

Overview of Siebel Open UI

This chapter describes an overview of Oracle's Siebel Open UI. It includes the following topics:

- [About Siebel Open UI on page 17](#)
- [Differences Between High Interactivity and Siebel Open UI on page 21](#)
- [About Using This Book on page 30](#)

About Siebel Open UI

This topic describes Siebel Open UI. It includes the following information:

- [Overview of Siebel Open UI on page 17](#)
- [Open Development Environment on page 19](#)
- [Multiple Client Environment on page 20](#)
- [Support for More Than One Usage on page 20](#)
- [New Notification User Interfaces on page 21](#)
- [Mobile Environments on page 21](#)

Overview of Siebel Open UI

Siebel Open UI is an open architecture that you can use to customize the user interface that your enterprise uses to display business process information. These processes must meet the requirements of a wide range of employee, partner, and customer applications. You can use Siebel Tools to do these customizations, and you can also use Web standards, such as HTML, CSS, or JavaScript. Siebel Open UI uses these standards to render the Siebel Open UI client in the Web browser. It uses no proprietary technologies, such as browser plug-ins or ActiveX.

Siebel Open UI can run any Siebel business application on any Web browser that is compliant with the World Wide Web Consortium (W3C) standards. It can display data in Web browsers that support Web standards on various operating systems, such as Windows, Mac OS, or Linux. For example:

- Internet Explorer
- Google Chrome
- Mozilla Firefox
- Apple Safari

Siebel Open UI uses current Web design principles, such as semantic HTML and unobtrusive JavaScript. These principles make sure configuration for the following items remains separate from one another:

- Data and metadata that determines HTML content
- Cascading Style Sheet configurations that determine styling and layout
- JavaScript behavior that determines interactivity and client logic

You can modify each of these items separately and independently of each other. For example, you can configure Siebel Open UI to hide some of the objects that it displays on a Siebel screen when it displays Siebel CRM data in a list or form on the smaller footprint of a mobile device. Hiding these objects, such as menus or tabs, can help to optimize mobile screen usage. Siebel Open UI can use swipe and zoom features that are native on a tablet for the same user interface that it uses for keyboard and mouse events that are native on a desktop.

Siebel Open UI can reference a third-party resource. For example, you can configure Siebel Open UI to get data from a supplier Web site, incorporate it with Siebel CRM data, and then display this data in the client. For example, it can get literature information from a supplier, and then include this information in a detailed display that includes information about the product, such as images, diagrams, or parts lists. It can mix this information with Siebel CRM data, such as customers who own this product, or opportunities who might be interested in purchasing this product.

The architecture that Siebel Open UI uses includes well-defined customization points and a JavaScript API that allow for a wide range of customization for styling, layout, and user interface design. For more information, see [“Architecture of Siebel Open UI” on page 37](#). For more information about the JavaScript API that Siebel Open UI uses, see [Appendix A, “Siebel Open UI Application Programming Interface.”](#)

For information about deploying Siebel Open UI, including supported features, see Article ID 1499842.1 on My Oracle Support. For more information about using Siebel Tools, see *Using Siebel Tools*.

Customizations That You Can Make with Siebel Open UI

The following list describes a few of the example customizations that you can make with Siebel Open UI. You can use JavaScript to implement most of these examples. It is often not necessary to use Siebel Tools to do these customizations:

- Refresh only the part of the screen that Siebel Open UI modifies.
- Use JavaScript to display and hide fields or to configure a spell checker.
- Configure Siebel Open UI to display a list applet as a box list, carousel, or grid.
- Display data from an external application in a Siebel CRM view or applet.
- Display a Siebel CRM view or applet in an external application.
- Configure Siebel Open UI to display a Google map.
- Use cascading style sheets to modify HTML elements, including position, dimension, and text attributes of an element.
- Use HTML to customize the logo that your company uses or to customize the background image.

The following are a few examples of customizations that you can make with Siebel Open UI in a mobile environment:

- Use JavaScript to configure menus, menu items, and the layout for mobile devices.

- Display Siebel CRM data in a Google map or add maps that include location data.
- Create a custom mobile list.
- Configure scrolling, swipe, swipe scrolling, infinite scrolling, and the height of the scroll area.
- Configure a view to use landscape or portrait layout.
- Configure toggle controls and toggle row visibility.
- Display a high interactivity view.

For more information about these examples, see [Chapter 5, “Customizing Siebel Open UI”](#) and [Chapter 9, “Customizing Siebel Open UI for Siebel Mobile.”](#)

Open Development Environment

You can use Siebel Tools or a development tool of your choice to customize Siebel Open UI so that it fits in your business environment and meets specific user requirements. You might not require Web development in many situations because the Siebel Tools configuration works for the Siebel Open UI client similarly to how it works for the current Siebel Web Client. You can use a predefined, uncustomized deployment, or you can use Siebel Tools to customize the SRF and Siebel Web templates. You can use only Web development or you can use Siebel Tools and Web development depending on the implementation requirements.

You can use Siebel Open UI with the rendering environment of your choice. You can use your preferred Integrated Development Environment (IDE) to write native JavaScript code on top of the API that Siebel CRM uses, or with the JavaScript API that Siebel Open UI uses. For more information, see [Chapter 5, “Customizing Siebel Open UI.”](#) For more information about the JavaScript API that Siebel Open UI uses, see [Appendix A, “Siebel Open UI Application Programming Interface.”](#)

You can use HTML, CSS, or JavaScript to add features. For example, you can do the following:

- Create smooth transitions between swipe, accordion, or carousel views.
- Create multifont displays.
- Expand, collapse, or resize an applet.
- Use open-source JavaScript code that can reuse work from the open-source development community.
- Use your preferred JavaScript environment, or use the environment that Siebel Open UI provides.
- Use a plug-in, proprietary development environment, or native development environment that you choose to create a custom rendering architecture that resides top of the JavaScript API that Siebel Open UI uses.
- Use intraworkspace communication and DOM (Document Object Model) access and manipulation through standard JavaScript programming.
- Do a limited pilot test of your customizations in your current Siebel Server implementation while most of your users continue to use the high-interactivity client.
- Preserve your existing customizations.

Siebel Open UI JavaScript API Support

The JavaScript API that Siebel Open UI uses replaces browser scripting. You can use your own Integrated Development Environment to write JavaScript and you can customize the JavaScript API that Siebel Open UI provides. This JavaScript API allows you to do the following:

- Include Siebel Open UI or individual Siebel Open UI objects, such as views or applets, in a third-party user interface.
- Integrate external content in the Siebel Open UI client.
- Use public and documented JavaScript APIs that support your business logic without rendering objects that depend on a specific or proprietary technology.

For more information about this JavaScript API, see [Appendix A, "Siebel Open UI Application Programming Interface."](#)

Multiple Client Environment

Siebel Open UI can do the following to support different client environments:

- Display data in any client that meets the World Wide Web Consortium standards. For example, a corporate desktop, laptop, seven-inch tablet, or ten-inch tablet. Siebel Open UI can display a typical Siebel CRM desktop client in the smaller footprint that a tablet provides.
- Display data in a browser or any other compliant client or mobile platform.
- Display data simultaneously from a single Siebel business application to more than one client environment.

Siebel Open UI works the same way for the following client types:

- Siebel Web Client
- Siebel Mobile Web Client
- Siebel Dedicated Web Client (Thick Client)

Support for More Than One Usage

Siebel Open UI adjusts to the unique attributes of each client so that the user can do the same task on a variety of client types. It can optimize the intrinsic capabilities of each client type or device so that they provide a desirable user experience for the novice or expert user. An administrator can also configure Siebel Open UI to meet these individual skill levels. Siebel Open UI can do the following:

- Support applications that you can customize to meet appearance and behavior requirements or usage patterns of various devices, such as smartphones, tablets, desktop computers, or laptop computers.
- Use flexible layout options that support a tree tab layout or a custom navigation design.
- Automatically hide tabs and navigation panes when not in use to optimize space.
- Allow employees, partners, and customers to use the same business process and validation with different levels of access.
- Use user interactions that are consistent with current Web applications.

- Support layout and gesture capabilities for mobile users who use a tablet or smartphone device.

New Notification User Interfaces

Siebel Open UI includes elements from social media and smartphones that improve user productivity, such as notification applets. It combines these capabilities with other Siebel CRM innovations to provide the following capabilities:

- Use a notification area that displays messages. The user can access this area at any time without disrupting current work.
- Hover the mouse to toggle between summary and detail information for a record.
- Use native Web browser functionality. For example, bookmarks, zoom, swipe, printing and print preview, and spelling checker.
- Use intuitive system indicators for busy events or to cancel a time-consuming operation.
- Allow navigation through a wide range of data entry and navigation capabilities through the keyboard, mouse, tablet, or gesturing.

For more information, see [“Notifications That Siebel Open UI Supports” on page 541](#).

Mobile Environments

Siebel Open UI on a mobile interface uses the same architecture that Siebel Open UI on a desktop application uses. For more information, see *Siebel Connected Mobile Applications Guide*.

Differences Between High Interactivity and Siebel Open UI

This topic describes the differences that exist between the high-interactivity client and the Siebel Open UI client. It includes the following information:

- [How Siebel CRM Renders High-Interactivity Clients on page 22](#)
- [How Siebel CRM Renders Siebel Open UI Clients on page 24](#)
- [Comparison of Customization Capabilities Between High Interactivity and Siebel Open UI on page 29](#)
- [Summary of Differences Between High Interactivity and Siebel Open UI on page 28](#)

How Siebel CRM Renders High-Interactivity Clients

The Siebel Server uses SWE (Siebel Web Engine) tags in SWE templates to create the screens that it displays in the high-interactivity client that a Siebel application uses. A *control* is a contained, user interface element, such as a menu, toolbar, grid, combo box, and so on. The red borders in [Figure 1](#) identify some of the controls that the Siebel Server renders.

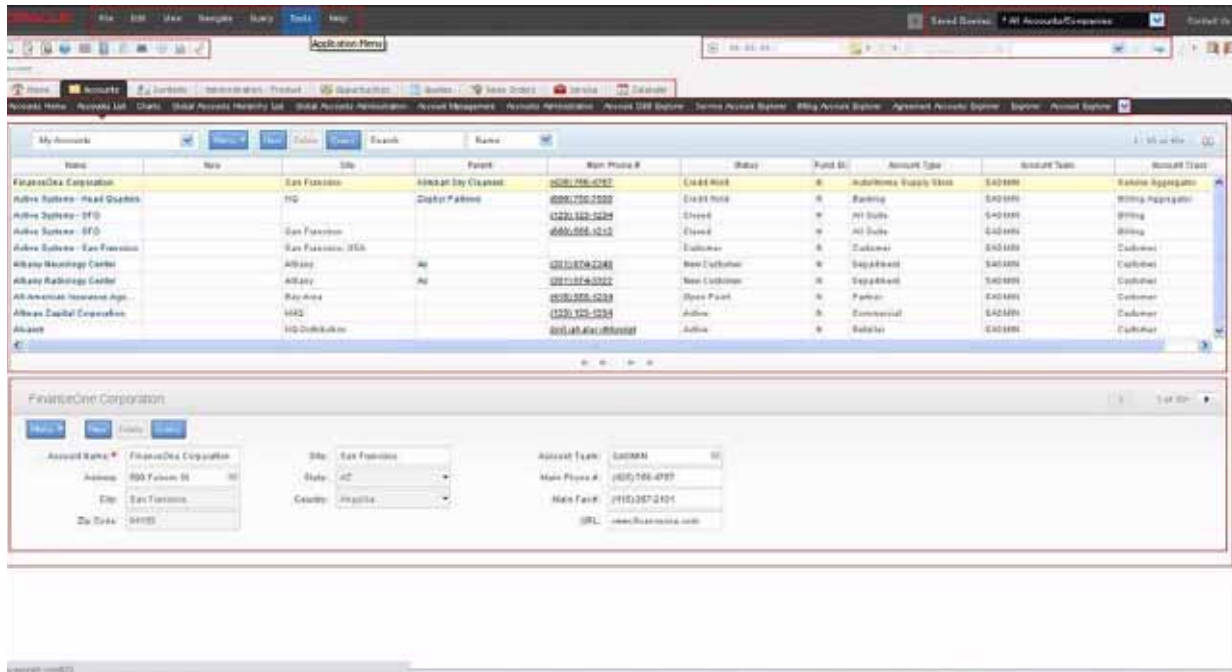


Figure 1. How Siebel CRM Renders High Interactivity Clients

A typical Siebel CRM Web page includes several controls that Siebel Web Template (SWT) files define. For example:

- Menus
- Toolbars
- Predefined query lists
- Screen tabs
- Applets

In high interactivity, each of these controls, or a group of controls, occupies an HTML frame. High interactivity positions the HTML frame and uses the position and dimension information that the HTML markup contains in this frame to hardcode them into place. High interactivity gets this information from the SWT files that it processes to render the page layout.

A *high interactivity client* is a type of Siebel CRM client that resembles a Windows client. It supports fewer browsers than standard interactivity, but it includes a set of features that simplify data entry. For example, page refreshes do not occur as often as they do in standard interactivity. The user can create new records in a list, save the data, and then continue browsing without encountering a page refresh. For more information about high interactivity and standard interactivity, see *Configuring Siebel Business Applications*.

How High Interactivity Rendering Affects Your Ability to Customize Siebel CRM

A SWE template allows you to customize a high-interactivity client only according to the capabilities that the SWE tags provide. For example, you can add a custom list applet to a view, but you cannot modify the individual objects that this list applet contains. You cannot modify a list applet to render as a carousel because a view web template can reference an applet, but it cannot reference the objects that the applet contains, and you cannot modify the ActiveX controls that do render these objects.

Figure 2 illustrates how Siebel CRM uses repository metadata to render objects in a high-interactivity client. For example, Siebel CRM uses:

- View metadata that resides in the repository on the Siebel Server to render a view object in the client.
- Applet metadata and Siebel CRM data that reside in the repository to render an applet object in the client.

Figure 2 illustrates how a standard, custom rendering capability is not available on the high interactivity client because Siebel CRM uses a SWE tag that it gets from the Siebel Server to render each control, and you cannot modify these tags in the client. The configuration on the client is for the most part a black box configuration. You cannot modify it, or it is difficult to modify objects in the client without using Siebel Tools to do custom binding.

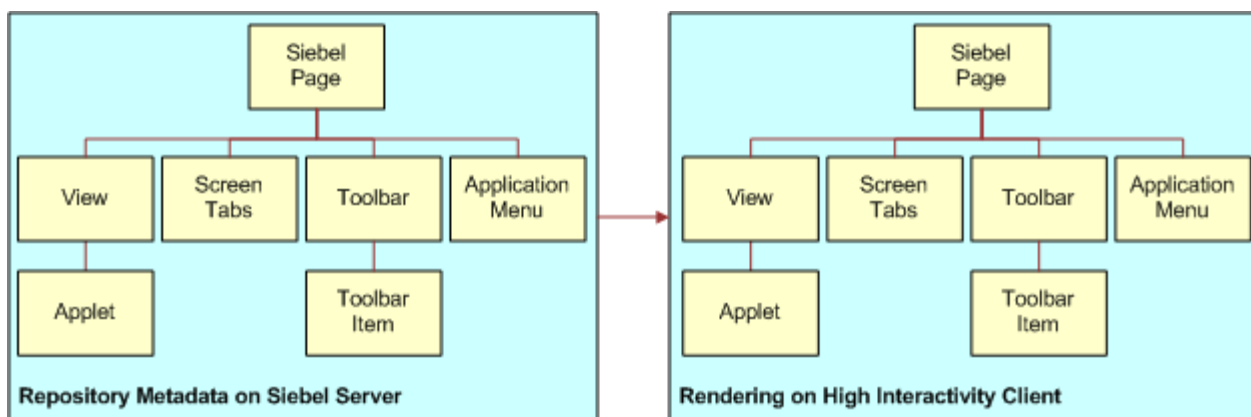


Figure 2. How Siebel CRM Uses Repository Metadata to Render Objects in High Interactivity Clients

Siebel CRM uses a SWE template to render each applet directly from the Siebel repository to the user interface in the client, and each applet references a business component to get Siebel CRM metadata and data from the repository. This configuration does not allow you to customize how Siebel CRM renders this applet unless you use Siebel Tools to modify the repository. For example, you cannot use JavaScript in the client to distribute data from a repository applet across more than one pane in a Siebel screen, such as displaying the address of a contact in a pane that is separate from the pane that displays other contact details, such as the contact name and phone number. You cannot use an alternative configuration, such as your custom configuration or a third-party configuration, to bind the Siebel business layer to user interface objects, except through Siebel Tools.

Only one view that displays content typically exists in a Siebel screen, and you cannot add more views unless you use Siebel Tools to modify the repository. Siebel Tools specifies the configuration for each instance of these objects that determines how Siebel CRM binds the object to the Siebel Business Layer. For example:

- To bind the command that a menu item or toolbar button calls
- To bind the business component and business component fields that an applet references to get Siebel CRM data

You can write scripts on the Siebel Server or the client, but these scripts only allow you to customize how Siebel CRM processes the requests that it receives from the user interface. They do not allow you to customize rendering.

For more information about applets, business components, views, the Siebel repository, Siebel metadata, Siebel Tools, and so on, see *Configuring Siebel Business Applications*

How Siebel CRM Renders Siebel Open UI Clients

Siebel CRM does the following to render a Siebel Open UI client:

- Uses HTML div elements and HTML tables in SWE templates to determine physical layout instead of the HTML frames that high interactivity uses. Siebel Open UI does not use div elements to structure a page. The entire page hierarchy that Siebel Open UI uses is a hierarchy of div elements. Siebel Open UI does not use the HTML frame.
- Uses cascading style sheets (CSS) to specify position, dimension, and styling for HTML elements, such as font color and font type, instead of the HTML code that high interactivity uses. This styling does not apply to the objects that an ActiveX control renders in a high-interactivity client, such as a list applet.

This configuration is more closely aligned with current guidelines for Web design than the configuration that high interactivity uses. Siebel Open UI allows you to customize how Siebel CRM renders individual objects in the client without having to use Siebel Tools, and it allows you use an alternative configuration, such as your custom configuration or a third-party configuration, to bind the Siebel business layer to user interface objects. Siebel Open UI allows you to customize an existing SWT file or create a new SWT file.

How Siebel CRM Renders Div Containers on Siebel Servers

Figure 3 illustrates how the Siebel Server uses SWE tags that reside in SWE templates to render div containers on the Siebel Server. For example, it renders a swe:view tag as a view container. It does the same rendering on this server for Siebel Open UI that it does for high interactivity.

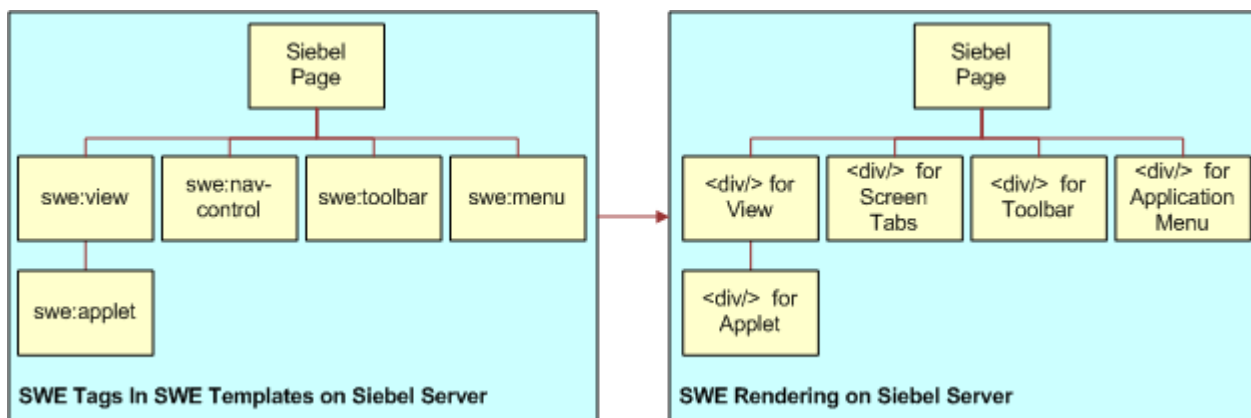


Figure 3. How Siebel Servers Use SWE Tags to Render Containers on the Siebel Server

How Siebel CRM Handles Data in Siebel Open UI

Figure 4 illustrates how Siebel CRM uses a *presentation model*, which is a JavaScript file that resides in the client that specifies how to handle the metadata and data that Siebel Open UI gets from the Siebel Server. Siebel CRM then displays this information in a list applet or form applet in the client. The presentation model provides a logical abstraction of the metadata, transaction data, and behavior for part of the user interface. Siebel Open UI includes a presentation model for each significant part of the user interface, such as the application menu, toolbars, screen tabs, visibility drop-down lists, applet menus, different types of applets, and so on. The presentation model does not render the HTML in the user interface.

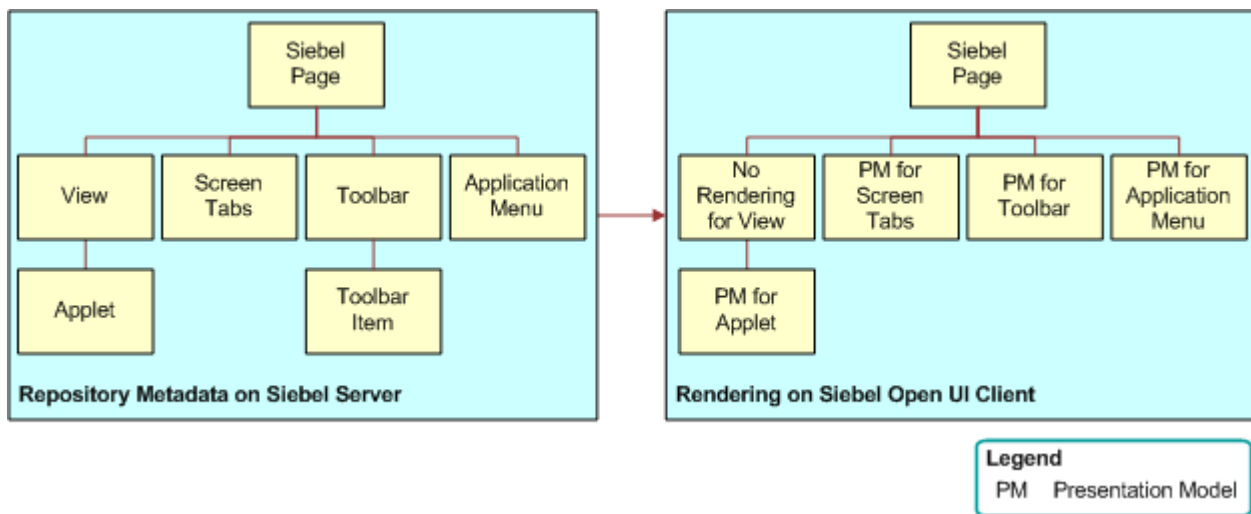


Figure 4. How Siebel CRM Handles Data in Siebel Open UI

How Siebel CRM Renders Objects in Siebel Open UI

Figure 5 illustrates how Siebel CRM uses a *physical renderer*, which is a JavaScript file that Siebel Open UI uses to build the user interface. A physical renderer contains instructions that describe how to render the physical presentation and interaction for a user interface element, such as a grid, carousel, form, tree, tab, menu, button, and so on. Each physical renderer references a presentation model, and it uses the metadata, data, and behavior that this presentation model defines to render an object in the client. For more information about presentation models and physical renders, see [“About the Siebel Open UI Development Architecture” on page 37](#).

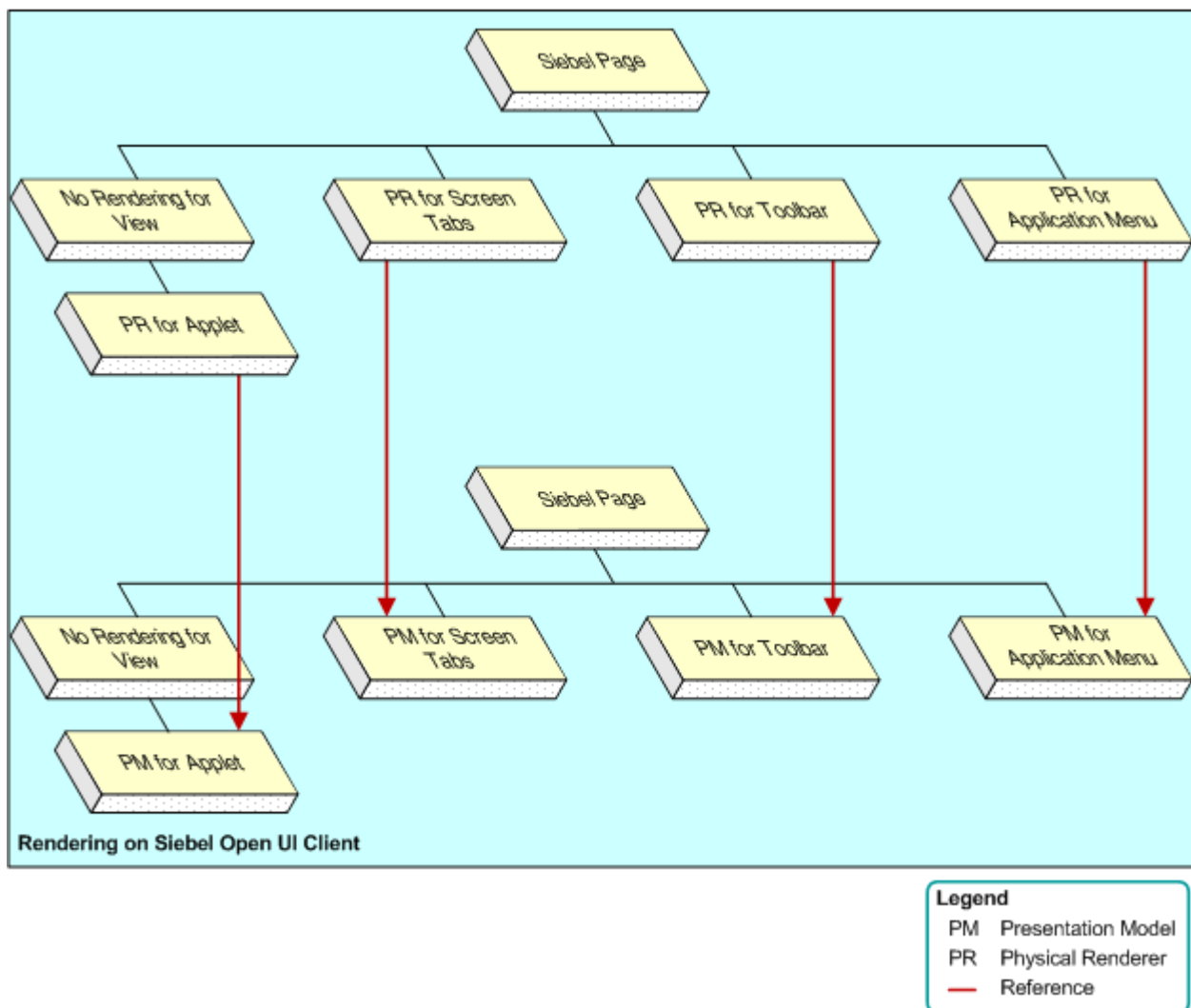


Figure 5. How Siebel CRM Renders Objects in Siebel Open UI

Examples of How You Can Customize Siebel Open UI

Siebel Open UI uses the presentation model and the physical renderer to separate the logical user interface from the rendering. This configuration allows you to modify the user interface without having to modify the logical structure and behavior of the client. For example, you can modify the physical renderer to use a third-party, grid-to-carousel control to display a list applet as a carousel without modifying a presentation model. For more information about this example, see [“Customizing List Applets to Render as a Carousel” on page 167](#).

You can use the physical renderer of a control to implement a variety of configurations so that Siebel Open UI can render this control at nearly any physical location in the browser and with your custom logic. You can use the physical renderer to display different parts of the same applet in different physical panes in a Siebel screen. For example, you can configure Siebel Open UI to display a temporary recycle bin that uses data from the presentation model to render data in a pane that is physically separate from the data that the list applet displays. For more information about this example, see [Chapter 4, “Example of Customizing Siebel Open UI.”](#)

You can use the presentation model to modify the logical behavior of the user interface without modifying the physical renderer. For example, you can modify a presentation model to add a list column in a list applet so that it iterates through list columns and renders them without modifying the physical renderer. This column can reside on the client even if the Siebel Server contains no representation of it.

Summary of Differences Between High Interactivity and Siebel Open UI

Siebel Open UI and high interactivity implement the physical user interface differently:

- **Siebel Open UI.** The SWE renderer that Siebel Open UI uses structures the physical user interface through HTML div elements instead of the HTML frames that high interactivity uses. This configuration allows you to use cascading style sheets to do the layout for these div elements instead of hard coding the position of HTML frames. It simplifies reconfiguring the HTML for different user interface themes.
- **High interactivity.** Uses a hierarchy of HTML frames. This hierarchy makes it difficult to reconfigure the rendered HTML to support different user interface themes.

Siebel Open UI uses the same browser proxy configuration that high interactivity uses. This structure includes the object type hierarchy for applications, views, applets, business objects, and business components. It implements each of these objects in JavaScript. These JavaScript objects render HTML and handle user interaction through DOM events. Siebel Open UI uses no ActiveX controls to render the physical user interface.

The applet object that resides in the proxy contains all the same metadata and data that it requires to render the physical user interface that high interactivity uses. For more information, see [“About Objects and Metadata” on page 33](#).

In some situations, Siebel Open UI gets more data locally from the proxy or the client, or it creates metadata in the client. A presentation model specifies how to display this local data. It also provides an interface to the proxy.

Siebel Open UI uses a physical renderer to implement the binding that exists between a predefined JavaScript control and the Siebel Object Model in the proxy.

To create the HTML DOM structure for a JavaScript control, the renderer uses the metadata and data that Siebel Open UI populates into the client proxy. The renderer uses methods that reside in the presentation model to access this metadata.

Comparison of Customization Capabilities Between High Interactivity and Siebel Open UI

High interactivity and Siebel Open UI share the following customization capabilities:

- Metadata configuration for all user interface objects resides in the Siebel Repository.
- Layout configuration of user interface objects resides in SWE templates.
- Negligible server scripting capabilities exist to render custom configurations.

[Table 4](#) summarizes some of the significant customization differences that exist between high interactivity and Siebel Open UI. For a more detailed comparison, see Article ID 1499842.1 on My Oracle Support.

Table 4. Summary of Customization Differences Between High Interactivity and Siebel Open UI

| High Interactivity | Siebel Open UI |
|---|---|
| Renders the user interface as a collection of HTML frames. | Renders the user interface as a collection of HTML div elements. |
| Uses hard coding to determine styling, sizing, and positioning. Limits the customizations you make to this style. | Uses CSS files to determine styling, sizing, and positioning. Allows you to fully modify the user interface. For example, you can create a custom user interface for a small, nondesktop environment. |
| Siebel Web Engine can only render an entire view. | Siebel Web Engine can render an entire view or only an individual applet. |
| Uses ActiveX. Limits the customizations that you can make in the user interface. | Uses JavaScript that allows you to fully customize how Siebel Open UI renders the user interface. Allows any current, compliant Web browser to use Siebel Open UI. |
| An applet can reference only a business component or a virtual business component. | An applet can reference a business service, business component, or virtual business component. |

Browser scripting for proxy objects resides in the client. For more information, see [“Browser Script Compatibility” on page 598](#).

About Using This Book

This topic includes information about how to use this book. It includes the following information:

- [“Important Terms and Concepts” on page 30](#)
- [“How This Book Indicates Computer Code and Variables” on page 31](#)
- [“How This Book Describes Objects” on page 32](#)
- [“About the Siebel Innovation Pack” on page 33](#)
- [“Support for Customizing Siebel Open UI” on page 33](#)
- [“Getting Help from Oracle” on page 35](#)

Important Terms and Concepts

This book uses the following terms and concepts that you must understand before you customize Siebel Open UI:

- A *user* is a person who uses the client of a Siebel business application to access Siebel CRM data.
- The *user interface* is the interface that the user uses in the client to access data that Siebel Open UI displays.
- The *client* is the client of a Siebel business application. Siebel Call Center is an example of a Siebel business application. Siebel Open UI renders the user interface in this client.
- The *server* is the Siebel Server, unless noted otherwise.
- An *administrator* is anyone who uses an administrative screen in the client to configure Siebel CRM. The Administration - Server Configuration screen is an example of an administrative screen.
- *Predefined Siebel Open UI* is the ready-to-use version of Siebel Open UI that Oracle provides to you before you make any customization to Siebel Open UI.
- A *Siebel CRM object* is an object that resides in the Siebel Repository File. For example, a screen, view, applet, business component, menu, or control is each an example of a Siebel object. A Siebel CRM applet is not equivalent to a Java applet. For more information, see *Configuring Siebel Business Applications*.
- A *predefined object* is an object that comes already defined with Siebel CRM and is ready to use with no modification. The objects that Siebel Tools displays in the Object List Editor immediately after you install Siebel Tools, and the objects that the SRF (Siebel Repository File) contains before you make any customization are predefined objects.
- A *custom object* is a predefined object that you modified or a new object that you create.
- The term *focus* indicates the currently active object in the client. To indicate the object that is in focus, Siebel CRM typically sets the border of this object to a solid blue line.
- To *derive* a value is to use one more properties as input when calculating this value. For example, Siebel Open UI can derive the value of a physical renderer property from one or more other properties.

- The term *class* describes a JavaScript class. It does not describe the Siebel class object type, unless noted otherwise, or unless described in the context of the Siebel Object Hierarchy. For more information about the Siebel class object type, see *Siebel Object Types Reference*.
- The term *reference* describes a relationship that exists between two objects, where one object gets information from another object or sends information to this object. For example, in the Siebel Object Hierarchy, the Opportunity List Applet references the Opportunity business component to get opportunity records from this business component, and the Opportunity business component references the S_OPTY table to get opportunity records from this table.
- The term *instance* describes the current, run-time state of an object. For example, a *business component instance* is a run-time occurrence of a business component. It includes all the run-time data that the business component currently contains, such as the values for all fields and properties of this business component. For example, an instance of the Contact business component includes the current, run-time value of the City field that resides in this business component, such as San Francisco. You can configure Siebel Open UI to get a business component instance, and then modify this data or call the methods that this business component references.

For more information about these terms and other background information, see the following items:

- A complete list of terms that this book uses, see [“Glossary” on page 607](#).
- Using the Siebel Open UI client, see *Siebel Fundamentals for Open UI*.
- Enabling the Siebel Server to run Siebel Open UI, see *Siebel Installation Guide* for the operating system you are using.
- Using Siebel Tools, see *Using Siebel Tools*.

How This Book Indicates Computer Code and Variables

Computer font indicates a value you enter or text that Siebel CRM displays. For example:

This is computer font

Italic text indicates a variable value. For example, the *n* and the *method_name* in the following format description are variables:

Named Method *n*: *method_name*

The following is an example of this code:

Named Method 2: WriteRecord

How This Book Indicates Code That You Can Use as a Variable and Literal

You can write some code as a literal or a variable. For example, the Home method sets a record in the current set of records as the active row. It uses the following syntax:

```
busComp.Home();
```

where:

- busComp identifies the business component that contains the record that Home sets.

You can use busComp as a literal or a variable. If you declare busComp as a variable in some other section of code, and if it contains a value of Account when you use the Home method, then Home sets a record in the Account business component as the active record. You can also use the following code, which also sets a record in the Account business component as the active record:

```
Account.Home();
```

Case Sensitivity in Code Examples

The code examples in this book use standard JavaScript and HTML format for uppercase and lowercase characters. It is recommended that you use the following case sensitivity rules that this book uses in code examples:

- All code that occurs outside of a set of double quotation marks (" ") is case sensitive. The only exception to this rule occurs with path and file names.
- All code that occurs inside a set of angle brackets (<>) is case sensitive. The only exception to this rule is any code that you enclose with a set of double quotation marks that you nest inside a set of angle brackets.

The following example is valid:

```
function RecycleBINPMModel () {  
    SiebelAppFacade.RecycleBINPMModel.superclass.constructor.apply(this, arguments);  
}
```

The following example is not valid. Bold font indicates the code that is not valid:

```
function RecycleBINPMModel () {  
    SiebelAppFacade.RecycleBINPMModel.superclass.constructor.apply(this, arguments);  
}
```

How This Book Describes Objects

For brevity, this book describes how an object, such as a user property, does something. For example, this book might state the following:

The Copy Contact user property copies contacts.

In strict technical terms, the Copy Contact user property only includes information that some other Siebel CRM object uses to copy contacts.

For brevity, to describe how Siebel CRM uses the value that a property contains, this book typically only describes the property name. For example, assume Siebel CRM displays the value that the Display Name property contains. This property is a property of a tree node object. This book only states the following:

Siebel CRM displays the Display Name property of the tree node.

In reality, Siebel CRM displays the value that the Display Name property contains.

About Objects and Metadata

A Siebel *object definition* defines the metadata that Siebel Open UI uses to run a Siebel application. The Account List Applet that Siebel Tools displays in the Object List Editor is an example of an object definition. It includes metadata that Siebel Open UI uses to render the Account List Applet, such as the height and width of all controls in the applet, and all the text labels that it must display on these controls. The *Siebel Repository* is a set of database tables that stores these object definitions. Examples of types of objects include applets, views, business components, and tables. You use Siebel Tools to create or modify an object definition.

The *object manager* hosts a Siebel application, providing the central processing for HTTP transactions, database data, and *metadata*, which is data that the object definitions contain. It is different from *Siebel CRM data*, which is data that is specific to your business, such as account names and account addresses.

For more information, *Configuring Siebel Business Applications*.

How This Book Describes Relationships Between Objects

An object definition includes properties and a property includes a value. For example, the Business Object property of the Account Address view contains a value of Account. To describe this relationship, this book might state the following:

The Account Address view references the Account business object.

Sometimes the relationship between objects occurs through more than one object. For brevity, this book does not always describe the entire extent of relationships that exists between objects through the entire Siebel object hierarchy. For example, because the Account business object references the Account business component, and the Account Address view references the Account business object, this book might state the following:

The Account Address view references the Account business component.

About the Siebel Innovation Pack

Oracle provides the functionality that this guide describes as part of Siebel Innovation Pack 2013. To use this functionality, you must install the innovation pack and do the postinstallation configuration tasks. For more information about the functionality that Siebel Innovation Pack 2013 includes, see the applicable *Siebel Maintenance Release Guide* on My Oracle Support.

Depending on the software configuration that you purchase, your Siebel business application might not include all the features that this book describes.

Support for Customizing Siebel Open UI

Siebel CRM supports the following customizations in Siebel Open UI. You must carefully consider the implications of doing this customization and development:

- Siebel Open UI allows you to use predefined or existing Siebel repository information in your deployment without customization. Siebel Open UI uses this repository information to render the user interface. This rendering does require user acceptance testing.
- You can use Siebel Tools to customize Siebel Open UI so that it works in your business environment and meets user requirements. Siebel Tools configuration for Siebel Open UI is similar to the configuration that you do for clients that Siebel CRM renders in high-interactivity and standard-interactivity. You configure the same Siebel Repository File and the same Siebel Web templates.
- You can use your Web development skills and the Siebel Open UI JavaScript API to customize Siebel Open UI. For details about this API, see [Appendix A, "Siebel Open UI Application Programming Interface."](#) Siebel Open UI uses this API to replace proprietary browser scripting that renders high-interactivity clients. Oracle continues to support browser scripting, but strongly recommends that you convert any browser script that your deployment currently uses so that it uses the Siebel Open UI JavaScript API.
- You can combine Siebel Tools development with development of the Siebel Open UI JavaScript API simultaneously, as needed.
- Siebel CMR supports including Siebel Open UI or individual Siebel Open UI objects in a third-party user interface. Views and applets are examples of Siebel Open UI objects.
- Siebel CMR supports integrating external content in the Siebel Open UI client.
- To rebrand your deployment and customize the user experience, you can modify the cascading style sheets that come predefined with Siebel Open UI.
- You can use HTML, CSS, or JavaScript to add features. For example, you can do the following:
 - Build user interfaces on any technology that can integrate with the Siebel Open UI JavaScript API.
 - Use your preferred, open-source JavaScript environment, such as jQuery, from the open-source development community, or you can use the environment that Siebel Open UI provides.
 - Use a plug-in, proprietary development environment, or a native development environment. You can use these environments to create a custom rendering architecture that integrates with the Siebel Open UI JavaScript API.
 - Use intraworkspace communication and DOM access and manipulation through JavaScript programming.
 - Do a pilot user acceptance test of your Siebel Open UI deployment that uses your current Siebel Server implementation. Users can continue to use the high interactivity Siebel client during this testing.
 - Preserve your existing configurations and customizations.

Support That Siebel Open UI Provides

It is strongly recommended that you carefully consider the support policies that this topic describes before you customize Siebel Open UI. For more information about the support that Oracle provides, see *Scope of Service for Siebel Configuration and Scripting - Siebel Open UI* (Article ID 1513378.1) on My Oracle Support.

Support for the Siebel Open UI JavaScript API

Oracle only supports usage and features of the Siebel Open UI JavaScript API as described in Oracle's published documentation. This policy makes sure that your deployment properly uses this API and helps to make sure your deployment works successfully. You are fully responsible for support of any custom code that you write that uses this API. For product issues that are related to this API, Oracle might request a minimal test case that exercises your API modifications.

Oracle supports your usage of an Integrated Development Environment (IDE) of your choice that you use to write native JavaScript code that you then deploy to work with the Siebel Open UI JavaScript API. Oracle does not support the features of or the quality of any third-party IDE.

Oracle supports your usage of the Siebel Open UI JavaScript API with a rendering environment and system integration that you choose. Oracle has implemented Siebel Open UI in HTML. You can use this implementation as a template for your deployment on other technologies. This template approach allows you to expedite development. However, Oracle can in no way support these customizations because this work is outside the scope of Oracle's support for customizations. It is recommended that you work with Oracle's Application Expert Services on any implementation issues you encounter that are related to the Siebel Open UI JavaScript API. For more information, see ["Getting Help from Oracle" on page 35](#).

If your current deployment includes an integration that resides on the desktop, and if this integration does not easily support migration to JavaScript integration, then it is recommended that you move this integration to the Siebel Server, or use a web service on the desktop that can integrate to this server.

Support for Code Suggestions, Examples, and Templates

Oracle provides code examples only to help you understand how to use the Siebel Open UI JavaScript API with Siebel Open UI. Oracle does not support your usage of these code examples. It only supports usage of this API as described in [Appendix A, "Siebel Open UI Application Programming Interface."](#)

Getting Help from Oracle

The predefined application that Oracle provides includes integration interfaces that allow you to modify or to create a new user interface. You can use these integration interfaces to create your own presentation model or physical renderer, at your discretion. It is your responsibility to create and maintain any customizations you make. For more information, see ["About the Presentation Model" on page 39](#) and ["About the Physical Renderer" on page 39](#).

To get help from Oracle with configuring Siebel Open UI, you can create a service request (SR) on My Oracle Support. Alternatively, you can phone Global Customer Support directly to create a service request or to get a status update on your current SR. Support phone numbers are listed on My Oracle Support. You can also contact your Oracle sales representative for Oracle Advanced Customer Services to request assistance from Oracle's Application Expert Services.

3

Architecture of Siebel Open UI

This chapter describes the architecture that you can use to customize Siebel Open UI. It includes the following topics:

- [About the Siebel Open UI Development Architecture on page 37](#)
- [Life Cycle of User Interface Elements on page 54](#)

About the Siebel Open UI Development Architecture

This topic describes the architecture that you can use to customize Siebel Open UI. It includes the following information:

- [Overview of the Siebel Open UI Development Architecture on page 37](#)
- [Example of How Siebel Open UI Renders a View or Applet on page 41](#)
- [Customizing the Presentation Model and Physical Renderer on page 43](#)
- [Stack That Siebel Open UI Uses to Render Objects on page 46](#)
- [Items in the Development Architecture You Can Modify on page 49](#)
- [Example Client Customizations on page 50](#)
- [Differences in the Server Architecture Between High Interactivity and Siebel Open UI on page 51](#)
- [Differences in the Client Architecture Between High Interactivity and Siebel Open UI on page 53](#)

Overview of the Siebel Open UI Development Architecture

Siebel Open UI uses objects to deploy each element that it displays in the client. You can customize each of these objects in a way that is similar to how you customize each object in a high-interactivity client. You can customize each object separately. Each object resides in a layer that implements a particular area of customization. For example, you can customize each of the following items that you can customize in high interactivity:

- Application
- Screen
- View
- Applet
- Menu

- Application menu
- Applet menu
- Toolbar
 - Application toolbar
- Navigation object
 - Tabs at different levels
 - Visibility menu
- Predefined Query (PDQ) menu

Architecture You Can Use to Customize Siebel Open UI

Figure 6 illustrates the basic architecture that you can use to customize Siebel Open UI.

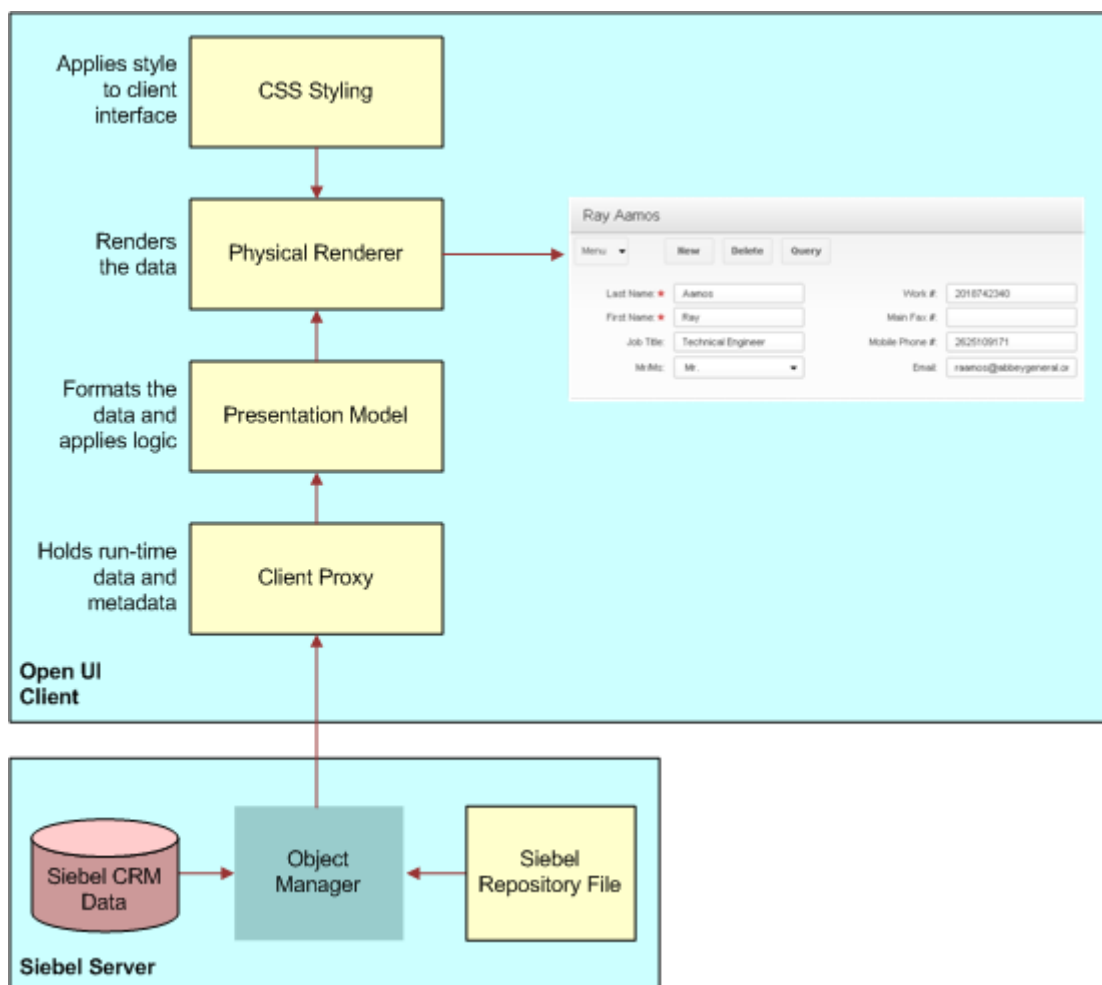


Figure 6. Architecture You Can Use to Customize Siebel Open UI

About the Presentation Model

A *presentation model* is a JavaScript file that specifies how to handle the metadata and data that Siebel Open UI gets from the Siebel Server, and then displays this information in a list applet or form applet in the client. It allows you to customize behavior, logic, and content. It determines the logic to apply, captures client interactions, such as the user leaving a control, collects field values, and sets properties. A presentation model can get the following items from the proxy, and then expose them for external use. These properties and methods are similar to the properties and methods that most software models use:

- **Properties.** Contains information about the current state of each user interface element. For example, if Siebel Open UI currently displays or hides a field.
- **Methods.** Implements behavior that modifies the state of an object. For example, if the user chooses a value, then a method can hide a field.

A presentation model can contain customization information that is separate from the predefined configuration information that Siebel Open UI uses for physical rendering. For example, it can display or hide a field according to a pick value.

For more information, see [“Example of a Presentation Model” on page 42](#).

About the Physical Renderer

A *physical renderer* is a JavaScript file that Siebel Open UI uses to build the user interface. It allows you to use custom or third-party JavaScript code to render the user interface. It binds a presentation model to a physical control. It can enable different behavior between a desktop client and a mobile client. It allows the presentation model to remain independent of the physical user interface objects layer. It can display the same records in the following different ways:

- List Applet
- Carousel
- Calendar
- Mind Map

For more information, see [“Example of a Physical Renderer” on page 43](#).

How Siebel Open UI Uses the Presentation Model and the Physical Renderer

A high-interactivity client allows you to use scripts, but it does not include a formal environment that binds data to the user interface. It customizes a controller instead of customizing a view. Siebel Open UI uses presentation models and physical renderers to meet this requirement.

A user interface object includes a combination of the following items:

- **Physical presentation and interaction for a user interface element.** For example, a grid, carousel, form, tree, tab, menu, button, and so on.

- **Logical presentation and interaction that Siebel Open UI can physically display in more than one way.** For example, Siebel Open UI can display a list of records in a grid or in a carousel. The logical representation of this data includes the metadata that Siebel Open UI uses to determine the Siebel CRM information that this list of records contains. It does not include information that Siebel Open UI uses to physically display this list as a grid or carousel.
- **Presentation and interaction information.** Includes application metadata, transaction data, and configuration information that determines client behavior. Siebel Open UI binds these items to the generic presentation. For example, it can determine whether or not a field is required, and then identify the data that it must display in a list column, or it can identify the business service method that it binds to a button.

A high-interactivity application can bind metadata, data, and logical behavior to a generic user interface in a highly configurable and declarative manner. It drives a fixed set of user interface presentation and interaction options. For example, you can configure a high-interactivity application so that a field is required or uses a hierarchical picklist. Siebel Open UI can use this configuration, but it also allows you to do the following customizations that you cannot do in high interactivity:

- **Add a completely new presentation or interaction feature in the user interface.** For example, display or hide a field according to a pick value.
- **Create a new or modify an existing logical user interface object.** For example, you can use Siebel Open UI to customize an object so that it displays a list of records in an *infinite scroll list*, which is an object that allows the user to view these records in a sliding window that displays records over a larger list of records that already exist in the client. It allows the user to do an infinite scroll in a mobile user interface. Note that, from a usability standpoint, it is almost always preferable to configure Siebel Open UI to use an interface that allows the user to page through sets of records rather than use a scroll list. This configuration reduces uncertainty regarding the records that Siebel Open UI has or has not displayed in the visible portion of the client.
- **Modify the type of user interface element that Siebel Open UI uses to display information.** For example, you can configure Siebel Open UI to display a list of records in a carousel instead of on a grid.

Example of How Siebel Open UI Renders a View or Applet

Figure 7 illustrates how Siebel Open UI renders the Contact Form Applet.

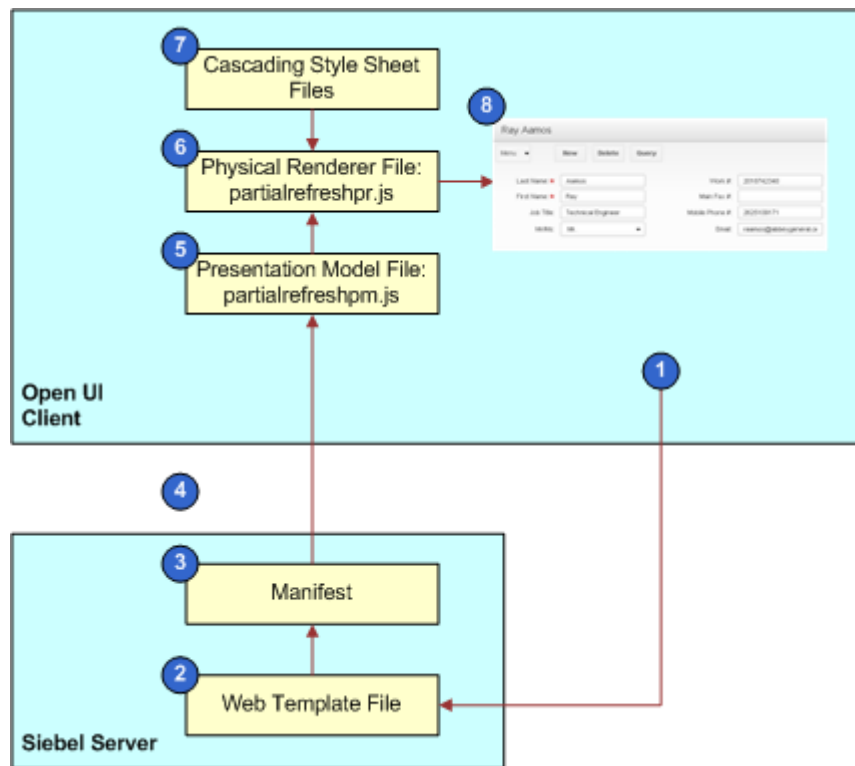


Figure 7. Example of How Siebel Open UI Renders a View or Applet

Explanation of Callouts

Siebel Open UI does the following to render the Contact Form Applet:

- 1 The user attempts to navigate to the Contact Form Applet.
- 2 Siebel Open UI creates the view that displays this applet. This creation is similar to how Siebel CRM creates a view in high-interactivity mode.
- 3 Siebel Open UI references the manifest to identify the files it must download to the client. For more information, see ["Configuring Manifests" on page 128](#).
- 4 Siebel Open UI downloads the JavaScript files it identified in [Step 3](#) to the client.
- 5 A presentation model formats the data and applies application logic. For more information, see ["Example of a Presentation Model" on page 42](#).
- 6 A physical renderer registers itself with a corresponding object. A presentation model also does this registration. For more information, see ["Example of a Physical Renderer" on page 43](#).

- 7 Siebel Open UI loads the cascading style sheets according to entries that the theme.js file contains.
- 8 Siebel Open UI uses a presentation model, physical renderer, and cascading style sheets to render the Contact Form Applet.

Example of a Presentation Model

Figure 6 describes how the partialrefreshpm.js file does a partial refresh. It is recommended that you include this business logic in a presentation model so that more than one modeler can reuse it. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. To view an example that uses this file, see [“Refreshing Applets That Contain Modified Data”](#) on page 159.

```

1 if( typeof( SiebelAppFacade.PartialRefreshPM ) === "undefined" ){
2   SiebelJS.Namespace( "SiebelAppFacade.PartialRefreshPM" );
3   define("siebel/custom/partialrefreshpm", [], function () {
4     SiebelAppFacade.PartialRefreshPM = ( function(){
5       function PartialRefreshPM( proxy ){
6         SiebelAppFacade.PartialRefreshPM.superclass.constructor.call( this, proxy );
7       }
8       SiebelJS.Extend( PartialRefreshPM, SiebelAppFacade.PresentationModel );
9
10      PartialRefreshPM.prototype.Init = function(){
11        SiebelAppFacade.PartialRefreshPM.superclass.Init.call( this );
12        this.AddProperty( "ShowJobTitleRelatedField", "" );
13        this.AddMethod( "ShowSelection", SelectionChange, { sequence : false, scope : this } );
14        this.AddMethod( "FieldChange", OnFieldChange, { sequence : false, scope : this } );
15      };
16
17      function SelectionChange(){
18        var controls = this.Get( "GetControls" );
19        var control = controls[ "JobTitle" ];
20        var value = this.ExecuteMethod( "GetFieldValue", control );
21        this.SetProperty( "ShowJobTitleRelatedField", ( value ? true: false ) );
22      }
23
24      function OnFieldChange( control, value ){
25        if( control.GetName() === "JobTitle" ){
26          this.SetProperty( "ShowJobTitleRelatedField", ( value ? true: false ) );
27        }
28      }
29    } )();
30  }
31 }

```

Figure 8. Example of a Presentation Model

Explanation of Callouts

The partialrefreshpm.js file includes the following sections:

- 1 Creates the JavaScript namespace.
- 2 Uses the Define method to make sure Siebel Open UI can identify the constructor. For more information, see [“Define Method”](#) on page 506.
- 3 Creates the presentation model class.
- 4 Customizes a predefined presentation model to support partial refresh logic.
- 5 Includes the logic that Siebel Open UI runs if the user changes records.
- 6 Includes the logic that Siebel Open UI runs if the user modifies a field value in a record.

Example of a Physical Renderer

Figure 9 describes how the partialrefreshpr.js file does a partial refresh for a physical renderer. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. To view an example that uses this file, see [“Refreshing Applets That Contain Modified Data”](#) on page 159.

```

1 if( typeof( SiebelAppFacade.PartialRefreshPR ) === "undefined" ){
  SiebelJS.Namespace( "SiebelAppFacade.PartialRefreshPR" );
2 define("siebel/custom/partialrefreshpr", ["order!3rdParty/jquery.signaturepad.min", "order!siebel/phyrenderer"], function () {
  SiebelAppFacade.PartialRefreshPR = { function(){

3      function PartialRefreshPR( pm ){
        SiebelAppFacade.PartialRefreshPR.superclass.constructor.call( this, pm );
      }

      SiebelJS.Extend( PartialRefreshPR, SiebelAppFacade.PhysicalRenderer );

      PartialRefreshPR.prototype.Init = function () {
        SiebelAppFacade.PartialRefreshPR.superclass.Init.call(this);
        this.AttachPMBinding( "ShowJobTitleRelatedField", ModifyLayout );
4      };

5      function ModifyLayout(){
        var controls = this.GetPM().Get( "GetControls" );
        var canShow = this.GetPM().Get( "ShowJobTitleRelatedField" );
        var WorkPhoneNum = controls[ "WorkPhoneNum" ];
        var FaxPhoneNum = controls[ "FaxPhoneNum" ];

        if( canShow ){
          $( "#div#WorkPhoneNum_Label" ).show();
          $( "[name=" + WorkPhoneNum.GetInputName() + "]" ).show();
          $( "#div#FaxPhoneNum_Label" ).show();
          $( "[name=" + FaxPhoneNum.GetInputName() + "]" ).show();
        }
        else{
          $( "#div#WorkPhoneNum_Label" ).hide();
          $( "[name=" + WorkPhoneNum.GetInputName() + "]" ).hide();
          $( "#div#FaxPhoneNum_Label" ).hide();
          $( "[name=" + FaxPhoneNum.GetInputName() + "]" ).hide();
        }
      }
    }
  }
}

```

Figure 9. Example of a Physical Renderer

Explanation of Callouts

The partialrefreshpr.js file includes the following sections:

- 1 Creates the JavaScript namespace.
- 2 Uses the Define method to make sure Siebel Open UI can identify the constructor. For more information, see [“Define Method”](#) on page 506.
- 3 Creates the physical renderer class.
- 4 Specifies the ShowJobTitleRelatedField property.
- 5 Includes the logic that Siebel Open UI runs if it modifies ShowJobTitleRelatedField.

Customizing the Presentation Model and Physical Renderer

Siebel Open UI uses two JavaScript files to implement the presentation model and the physical renderer that it uses to display an applet. For example, it uses the following files to display a carousel:

- ListPModel.js for the presentation model
- CarouselRenderer.js for the physical renderer

It uses the following files to display a grid:

- JQGridRenderer.js for the physical renderer
- ListPModel.js for the presentation model

Customizing the Presentation Model

Siebel Open UI considers static and dynamic values as part of the presentation model that it uses. For example, a list applet includes columns and renders data in each column in every row. Metadata specifies the column name and other details for each column, such as required, editable, and so on. These values are static. Siebel Open UI does not modify them unless you modify them as part of a customization effort. A list applet can also include dynamic values. For example, a value that identifies the record that is in focus, or the total number of visible records. Siebel Open UI can modify the value of a dynamic value in reply to an external event according to the behavior of the model. For example, if the user clicks a field in a record, and if this record is not in focus, then Siebel Open UI modifies the property that stores the focus information to the record that the user clicked. You can implement this type of functionality in a presentation model. For more information, see [“About the Presentation Model” on page 39](#).

Example of Customizing the Static and Dynamic Values of a Presentation Model

You can modify a presentation model to add a list column. For example, you can modify the SIS Product List Applet so that it displays a Select column that allows the user to choose more than one record, and then press Delete to delete them. You only modify a presentation model to implement this example. You do not modify a physical render. Siebel Open UI uses the JQGridRenderer physical renderer for the grid control. JQGridRenderer is sufficiently generic that it can iterate any list of columns that the presentation model returns. To view an example of this modification, see [“Customizing List Applets to Render as a Table” on page 176](#).

Example of Customizing the Behavior of a Presentation Model

You can add behavior to a presentation model. For example, you can configure a presentation model to display or hide a set of fields according to the value of another field. You can configure Siebel Open UI so that the Job Title field on the Contacts form applet determines whether or not it displays the Work# field and the Main Fax# field of a contact. If the Job Title includes a value, then Siebel Open UI displays the Work# field and the Main Fax# field. A presentation model controls this conditional display. The physical renderer requires no configuration to implement this example. It queries the presentation model, and then renders these fields according to the instructions that it gets from the presentation model. You can implement this behavior on the client without modifying any configuration on the Siebel Server. For a detailed description of an example that uses this type of configuration, see [Chapter 4, “Example of Customizing Siebel Open UI.”](#)

Customizing the Physical Renderer

You can use a physical renderer to modify how Siebel Open UI renders an object. For example, Siebel Open UI displays the predefined Contact Affiliations list applet as a typical Siebel CRM list. You can modify this list to display as a carousel. You can modify how the user scrolls through a set of records, which is a physical aspect of the applet that a physical renderer defines. But this list is still a list of records that is a logical representation of the applet that the presentation model defines. You do not modify this logical representation. To view an example of this type of modification, see [“Customizing List Applets to Render as a Carousel” on page 167](#). For more information, see [“About the Physical Renderer” on page 39](#).

Stack That Siebel Open UI Uses to Render Objects

Figure 10 describes the stack that Siebel Open UI uses to render objects. It uses the applet object as an example.

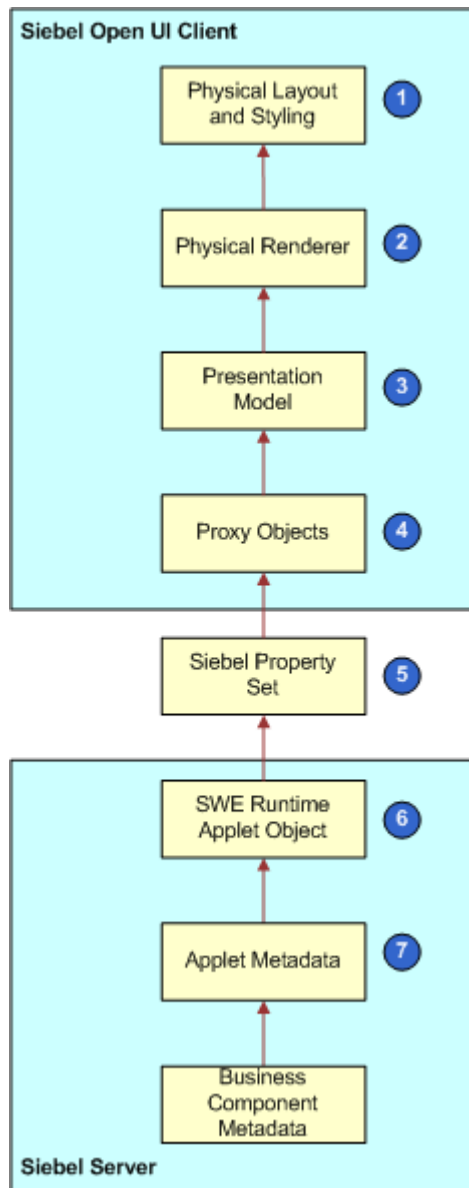


Figure 10. Stack That Siebel Open UI Uses to Render Objects

Explanation of Callouts

The stack that Siebel Open UI uses to render objects includes the following items:

- 1 **Physical layout and styling.** Allows you to use HTML to display content, JavaScript to control logic, and cascading style sheets to control layout and styling in the client. You can position or hide controls to achieve almost any layout requirement. This high level of customization is not possible with a high-interactivity client because high interactivity hard codes the physical layout.
- 2 **Physical renderer.** For more information, see [“About the Physical Renderer” on page 39.](#)
- 3 **Presentation model.** For more information, see [“About the Presentation Model” on page 39.](#)
- 4 **Proxy objects.** Includes object instances for the client proxy. Each of these instances represents an instance of a corresponding repository object that resides on the Siebel Server. Example objects include a view, applet, business object, or business component. A proxy object includes only enough logic to allow the client to use the same functionality that the server object uses, including the data and metadata that the server object requires. A proxy object exposes the interface for scripting in the client, but it does not allow you to significantly modify the physical user interface. You can only customize the flow of information from the Siebel Server to the client. You cannot customize how Siebel Open UI uses the metadata or data in the proxy object to render the physical user interface. In this example, proxy objects include the applet proxy and business component proxy that contain data and metadata from the Server Response property set. For more information, see [“Browser Script Compatibility” on page 598.](#)
- 5 **Siebel Property Set.** A hierarchy that Siebel Open UI uses to communicate between objects that reside on the Siebel Server and the proxies that reside in the client. The high-interactivity client uses the same format for this property set.
- 6 **SWE run-time applet object.** Exposes scripting interfaces that allow you to modify the applet so that it can control the business component or business service that this applet references. The applet that resides on the Siebel Server gets a request from the proxy applet instance that resides in the client. If necessary, it sends the request to a business component or business service. Siebel Open UI does not currently include a scripting interface that allows you to modify the property set that the applet sends to the client.
- 7 **Applet metadata.** The applet object in the Siebel Repository File (SRF) that contains information that Siebel Open UI uses to bind the user interface to the business component. Siebel Open UI maps this information through business component fields. This binding can include only a one-to-one mapping between one applet control and one business component field. Siebel Open UI does not allow more complex bindings. You can get data through a presentation model in the client to develop functionality that is similar to the functionality that more complex binding provides. For more information, see [“About Objects and Metadata” on page 33.](#)

Example Stack That Siebel Open UI Uses to Render Objects

This topic describes a typical example of how Siebel Open UI uses a presentation model and physical renderer for an applet that it displays in a view. Every object that Siebel Open UI renders uses this same object stack. You can customize objects in this stack to modify Siebel Open UI rendering and behavior. For example, you can customize the presentation model and physical renderers that implement view navigation to use tree navigation instead of the predefined nested tab navigation.

Figure 11 describes an example stack that Siebel Open UI uses to display a calendar applet.

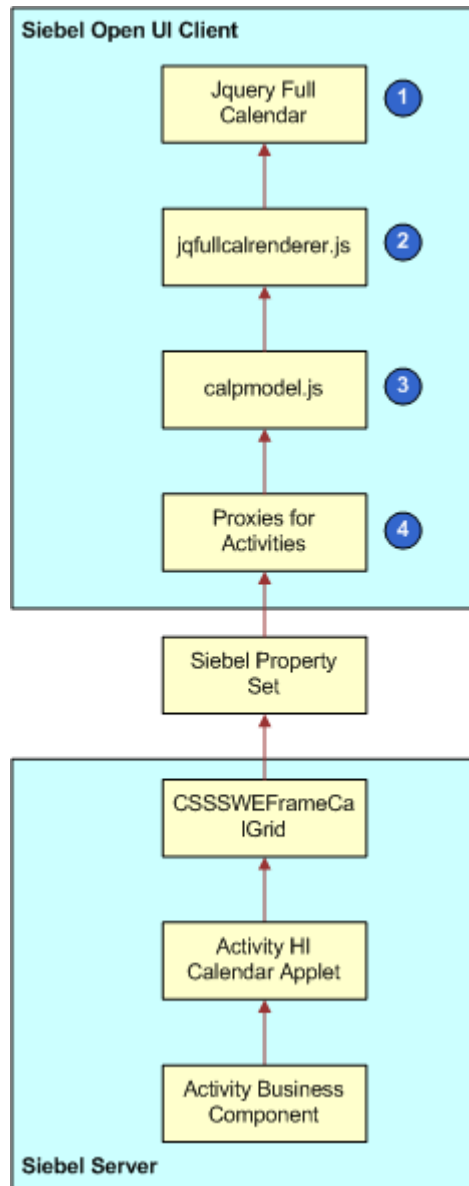


Figure 11. Example Stack That Siebel Open UI Uses to Render Objects

Explanation of Callouts

Siebel Open UI uses the following items to display a calendar applet:

- 1 **Jquery FullCalendar.** The physical JavaScript control. A third-party typically provides this control.

- 2 **jqfullcalrenderer.js**. Binds the CallPresentationModel object that the calpmodel.js file contains with the third-party calendar control.
- 3 **calpmodel.js**. Describes the logical behavior for the calendar user interface that Siebel Open UI displays on top of a list applet that runs in high interactivity.
- 4 **Activity proxies**. Includes proxies for the Activity HI Calendar Applet and the Activity business component.

Items in the Development Architecture You Can Modify

Figure 12 indicates the predefined items in the development architecture that Oracle provides and the items that you can modify. It delineates areas where you can customize Siebel Open UI.

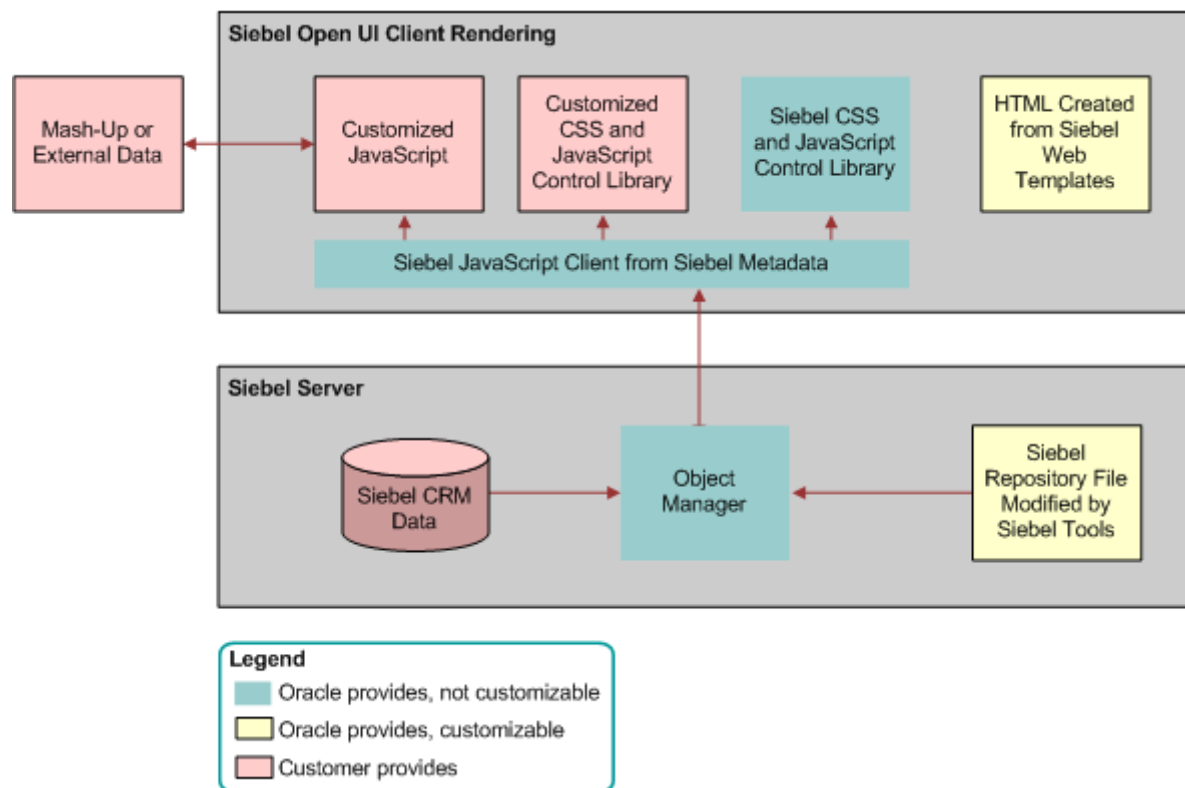


Figure 12. Items in the Development Architecture You Can Modify

Example Client Customizations

Table 5 describes some example client customizations you can do in Siebel Open UI. For detailed examples, see [Chapter 5, “Customizing Siebel Open UI.”](#)

Table 5. Example Client Customizations

| Customization | Work You Must Do |
|---|---|
| Customize a list applet or form applet. | You can use Siebel Tools to customize a list or form applet in the Siebel Repository. This work completes the basic binding to the Siebel object layer and displays a list or form in the client. No client customization is required. For more information, see <i>Using Siebel Tools</i> . |
| Add custom client behavior. | <p>You modify a presentation model. For example:</p> <ul style="list-style-type: none"> ■ Display or hide a control. For example, show a control if the user chooses a value from a drop down list. You add the required logic to a presentation model. You add or remove the control from the set of controls that Siebel Open UI already displays in the applet proxy in the client. For example, to add a local control in the client, you add this control in the presentation model to the set of controls that the proxy already contains. <p>Some configuration requirements do not require you to modify the physical renderer. For example, it is not necessary to modify the physical renderer to display a control because the predefined implementation for getting all fields from the client is already available.</p> <ul style="list-style-type: none"> ■ Modify the theme of a page. For example, you can configure Siebel Open UI to modify the theme of a page if the user modifies the orientation of a tablet device. You modify a state variable in a presentation model from portrait to landscape. You must modify the physical renderer for this example. You must add the logic that modifies styles that the user interface elements use when Siebel Open UI modifies the orientation state in the presentation model. |
| Add generic client behavior. | You use a physical control to render the presentation model. For example, to render a list applet as a carousel, you use the appropriate third-party control. |
| Position controls and customize style. | You modify CSS files. |

Differences in the Server Architecture Between High Interactivity and Siebel Open UI

Figure 13 compares the server architecture between high interactivity and Siebel Open UI.

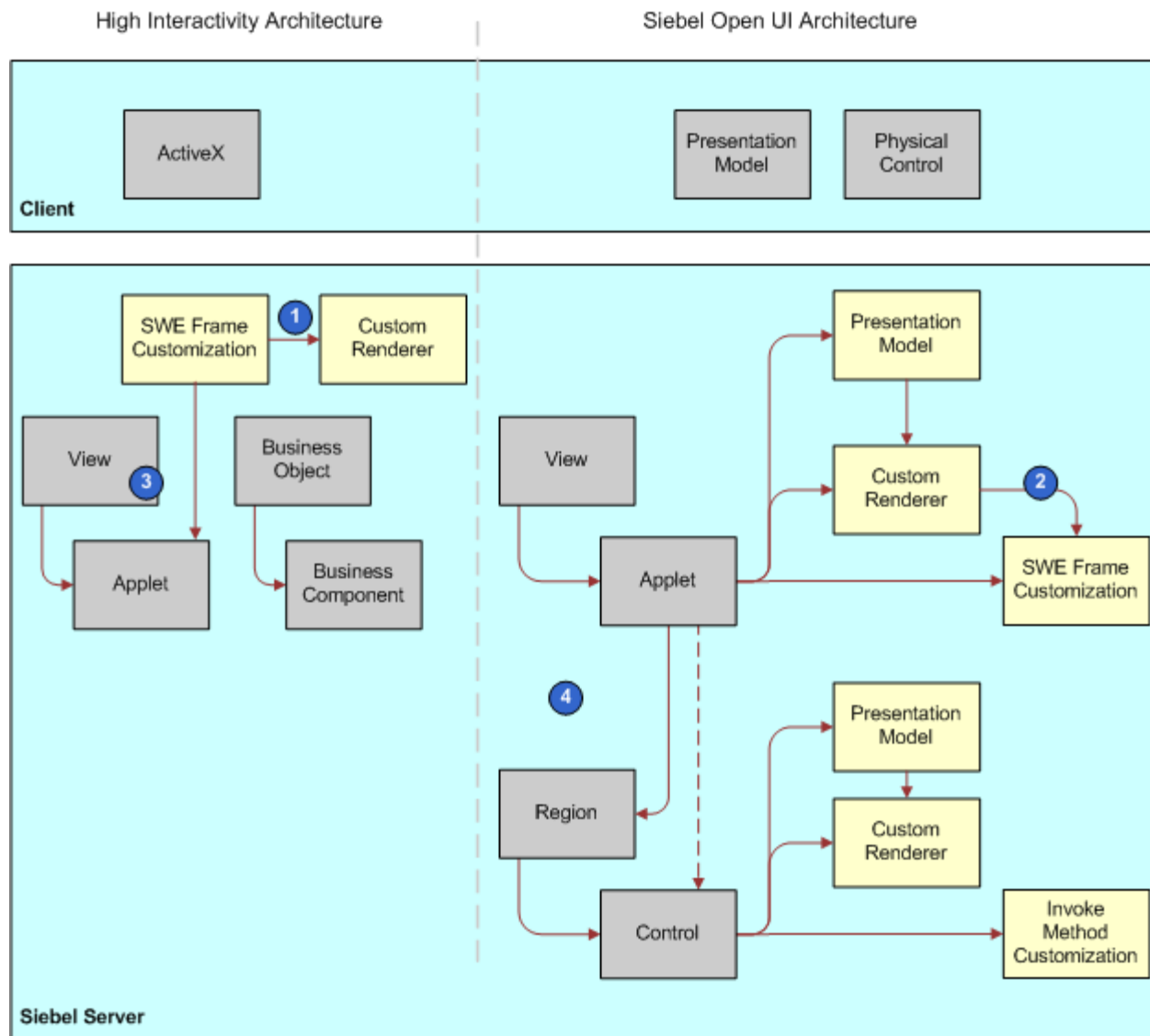


Figure 13. Comparing Server Architecture Between High Interactivity and Siebel Open UI

Explanation of Callouts

This comparison between the architecture that high interactivity uses and that Siebel Open UI uses includes the following items:

- 1 Rendering customization in high interactivity requires you to use a SWEFrame customization at the applet level.
- 2 Rendering customization in Siebel Open UI allows you to use SWEFrame customization, an equivalent customization, or to customize the physical renderer independently at any level of the object hierarchy, including at the subapplet level for an applet control.
- 3 High interactivity always starts rendering at the view level. It uses predefined code in the user interface hierarchy, from a request processing perspective.
- 4 Siebel Open UI uses objects, so rendering can occur at the screen, view, applet, or control level.

Differences in the Client Architecture Between High Interactivity and Siebel Open UI

Figure 14 compares the ActiveX UI architecture that a high-interactivity client uses to the architecture that Siebel Open UI uses.

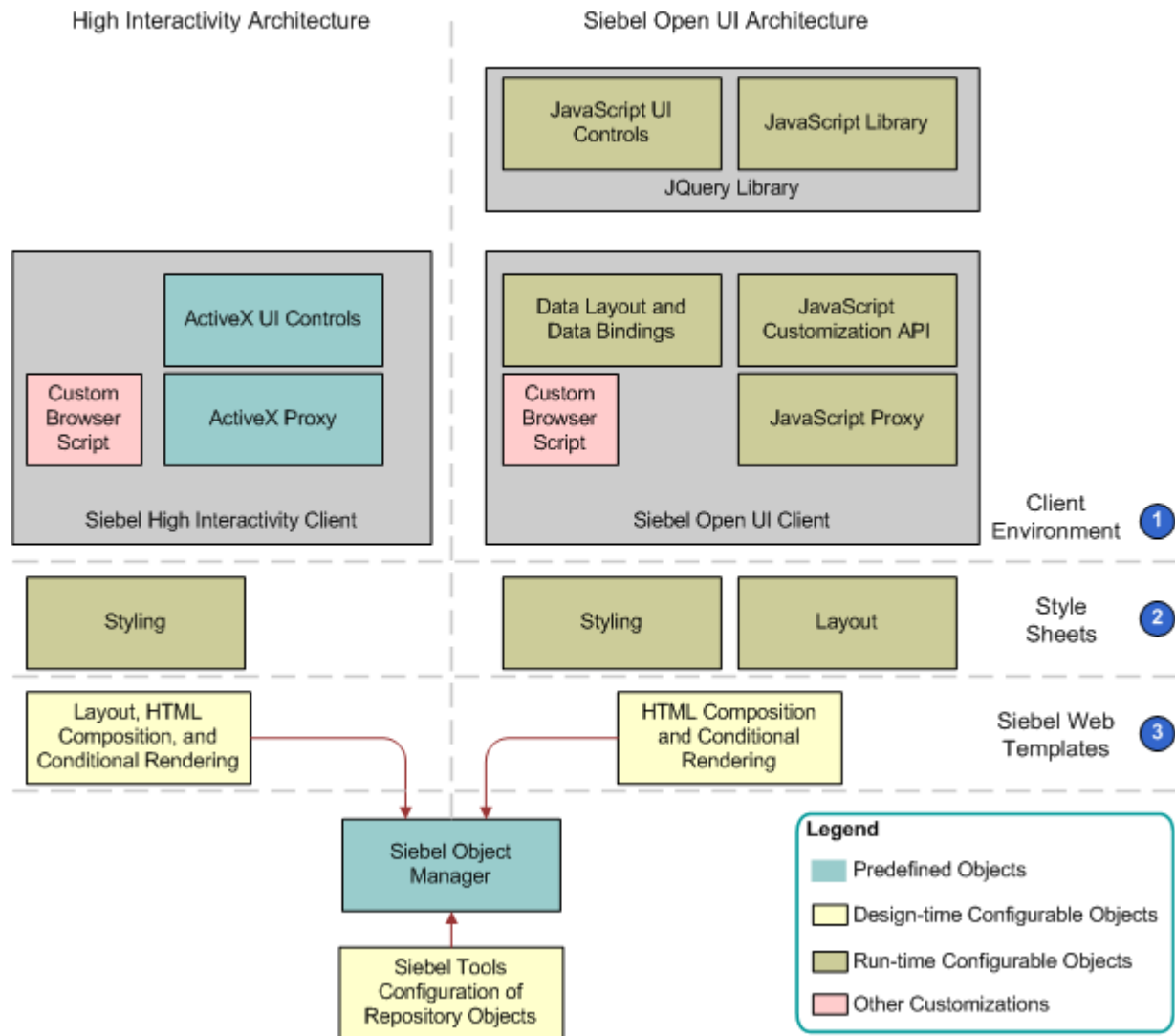


Figure 14. Comparing Client Architecture Between High Interactivity and Siebel Open UI

Explanation of Callouts

This comparison between high interactivity and Siebel Open UI includes the following items:

- 1 **Client Environment.** The Siebel Open UI client environment allows you to customize run-time configurable objects to meet a wide range of rendering requirements, from supporting more than one Web browser type to deploying to various client form factors.
- 2 **Style sheets.** The Siebel application or Web Server serves static style sheets.
- 3 **Siebel Web Templates.** The Siebel application or Web Server serves dynamic Siebel Web Templates.

Life Cycle of User Interface Elements

This topic describes how Siebel Open UI uses presentation model methods and physical renderer methods, and the methods that the presentation model and physical renderer calls during the life cycle of a user interface element.

The presentation model uses the following sequence of methods:

- 1 **Init**
- 2 **Setup**

The presentation model processes the events that it receives from the physical renderer during the life cycle. It also processes the replies for requests that the Siebel Server sends. Siebel Open UI can make the following calls to the presentation model during a life cycle:

- Call from the physical renderer because of a user action.
- Notification that the Siebel Server sends. For more information, see [“Notifications That Siebel Open UI Supports” on page 541](#).
- Reply property set that the Siebel Server sends.
- Completion request to get a follow-up request after the proxy finishes processing a reply from the Siebel Server.

The physical renderer continues to render each modification that occurs in the presentation model, and the AttachPMBinding method binds each of these modifications during the Init call to the physical renderer. One of the following items then signals these modifications:

- Siebel Open UI runs a presentation model method.
- Siebel Open UI modifies the value of a presentation model property.

For more information about the methods that this topic describes, see [Appendix A, “Siebel Open UI Application Programming Interface.”](#)

Summary of Presentation Model Methods

This topic summarizes some of the methods that a presentation model uses during the life cycle of a user interface element.

How Siebel Open UI Uses the Init Method of the Presentation Model

The Init method uses the following methods to configure the properties, methods, and bindings of the presentation model. For an example that uses Init, see [“Creating the Presentation Model” on page 62](#):

- **AddProperty.** Adds a property to a presentation model. This property can be simple or derived. If you use AddProperty to define a derived property, then Siebel Open UI uses the Get method on the presentation model to calculate and return the property value. For more information about deriving values, see [“About Using This Book” on page 30](#). For more information, see [“Get Method” on page 431](#).
- **AddMethod.** Adds a method to the presentation model. For more information, see [“AddMethod Method” on page 425](#).
- **AttachEventHandler.** Attaches a method that handles the logical event. Siebel Open UI calls this method when it sends an event to the presentation model through the OnControlEvent method. For more information, see [“OnControlEvent Method” on page 432](#) and [“AttachEventHandler Method” on page 427](#).
- **AttachNotificationHandler.** Attaches a method that handles the notification that Siebel Open UI calls when the Siebel Server sends a notification to an applet. A *notification* is a message that Siebel Open UI sends to the client when this client requests Siebel Open UI to modify a business component. For example, to create or delete a business component record. For more information, see [“Notifications That Siebel Open UI Supports” on page 541](#).
- **AttachPSHandler.** Handles other incoming property sets that the Siebel Server sends to the client. It can extract the values that a property set contains to individual properties or do other processing.
- **AttachPreProxyExecuteBinding.** Attaches a method to the presentation model. Siebel Open UI calls AttachPreProxyExecuteBinding before it processes the reply that it receives from the Siebel Server, but after it receives a reply from this server to the method that Siebel Open UI supplies as an argument. For more information, see [“Customizing Events” on page 118](#).
- **AttachPostProxyExecuteBinding.** Attaches a method to the presentation model. Siebel Open UI calls AttachPostProxyExecuteBinding after it processes the reply from the Siebel Server.

The physical renderer calls the following presentation model methods:

- **Get.** Gets the value of a property that resides in a presentation model.
- **ExecuteMethod.** Runs a method that the AddMethod method calls. For more information, see [“ExecuteMethod Method” on page 430](#).
- **OnControlEvent.** Calls an event. The physical renderer uses the OnControlEvent method to call the presentation model and send an event. The presentation model uses a binding that exists between the event and the presentation model method and the AttachEventHandler method to call the method. For more information, see [“OnControlEvent Method” on page 432](#) and [“AttachEventHandler Method” on page 427](#).
- **SetProperty.** Sets the value of a presentation model property. The physical renderer can set this value directly in some situations. For more information, see [“SetProperty Method” on page 432](#).

How Siebel Open UI Uses the Setup Method of the Presentation Model

The Setup method extracts the values that a property set contains. If Siebel Open UI creates an object on the Siebel Server, such as a frame, then this server sends the property set that describes this object to the client. Siebel Open UI uses this property set to set up the presentation model properties in the client. The Setup method uses the AddProperty method to extract this property set into presentation model properties. It does this work the first time Siebel Open UI creates the user interface object in the client. For more information, see [“Methods That Manipulate Property Sets” on page 518](#). For an example that uses Setup, see [“Customizing the Setup Logic of the Presentation Model” on page 64](#).

Life Cycle of a Physical Renderer

Figure 15 illustrates the life cycle of a physical renderer. For examples of various life cycle flows, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

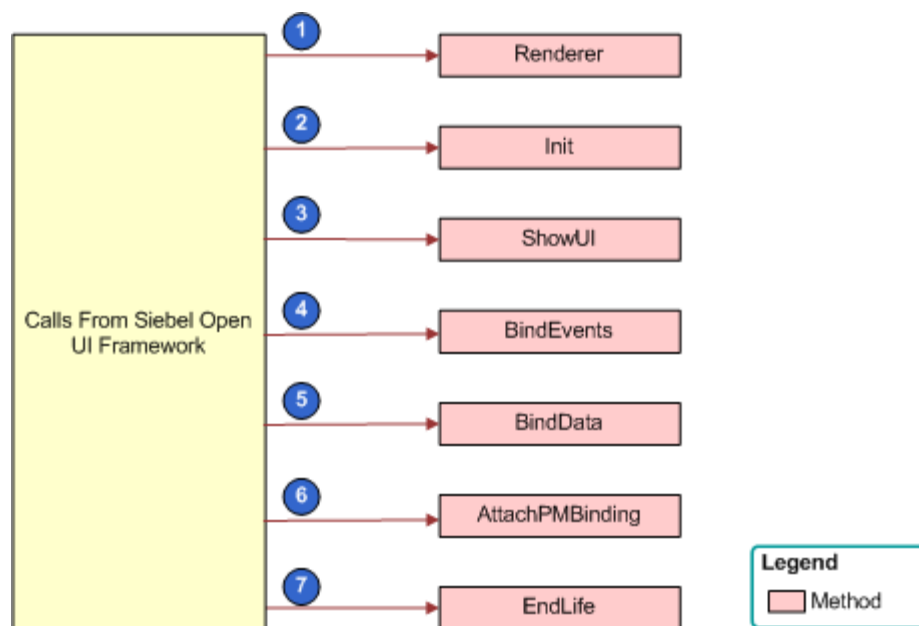


Figure 15. Life Cycle of a Physical Renderer

Explanation of Callouts

The physical renderer uses methods in the following sequence:

- 1 Renderer.** Creates the renderer.
- 2 Init.** Initializes and sets up the AttachPMBinding method. For more information, see [“Init Method” on page 431](#).

- 3 **ShowUI.** Displays the physical control that corresponds to an applet control. It renders the container for the metadata, data, and physical event bindings. For example, when Siebel Open UI renders a list applet as a grid, ShowUI renders the third-party grid control that it uses for the applet. For more information, see [“ShowUI Method” on page 464](#).
- 4 **BindEvents.** Sets up the user interface binding of physical events to the physical user interface, represented as HTML elements. It captures the user actions, and then translates these actions to logical events in the physical renderer before Siebel Open UI sends them to the presentation model for processing. For more information, see [“BindEvents Method” on page 461](#).
- 5 **BindData.** Downloads metadata and data from the Siebel Server to the client proxy, and then binds this data to the user interface. The list columns that a list applet uses is an example of metadata, and the record set that this list applet uses is an example of data. For more information, see [“BindData Method” on page 461](#).
- 6 **AttachPMBinding.** Attaches handlers to notifications that occur during the life cycle. For more information, see [“AttachPMBinding Method” on page 428](#). For more information about notifications that can occur during the life cycle, see [“Notifications That Siebel Open UI Supports” on page 541](#).

GetPM. Calls a method that the presentation model contains. It is recommended that you use GetPM only to call the following presentation model methods:

- ExecuteMethod
- OnControlEvent
- Get
- SetProperty

You can use ExecuteMethod or OnControlEvent to call a method that modifies the state of the presentation model or to call a method that reads this state. You can use the Get method to get the value of a presentation model property. You can use SetProperty to set the value of a presentation model property.

For more information, see [“GetPM Method for Physical Renderers” on page 463](#) and [“OnControlEvent Method” on page 432](#).

- 7 **EndLife.** Ends the life of the physical renderer. For more information, see [“EndLife Method” on page 462](#).

Example of the Life Cycle of a User Interface Element

Figure 16 describes the life cycle of the calendar user interface element.

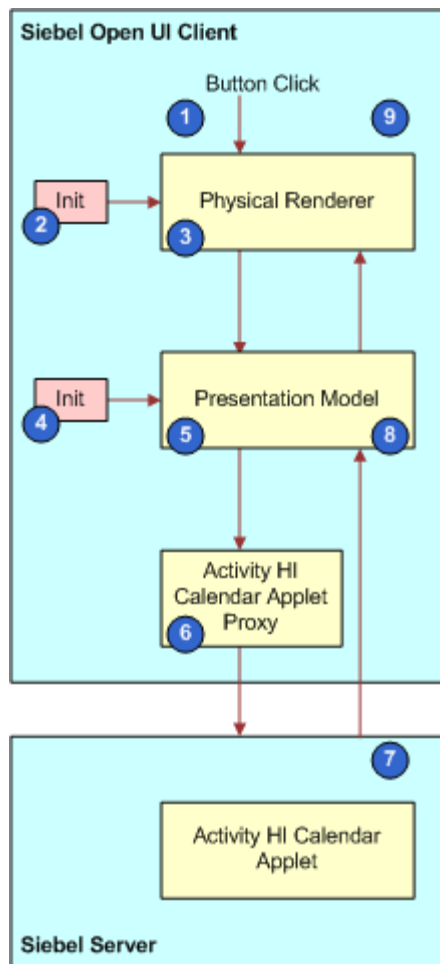


Figure 16. Example of the Life Cycle of a User Interface Element

Explanation of Callouts

The following sequence occurs during the life cycle of a calendar user interface object:

- 1 The user clicks a button that refreshes the calendar.
- 2 The Init method adds the following items to the physical renderer:
`AttachPMBinding("ProcessCalendarData", RefreshUI)`
- 3 The physical renderer sends the following method to the presentation model:
`OnControlEvents("Refresh_Calendar", RequestCalendarData)`
 For more information, see ["OnControlEvents Method" on page 432](#).

- 4 The Init method adds the following items to the presentation model:

```
AddProperty (MeetingDates, list of dates)
AddMethod (RequestCalendarData, implementation)
AttachEventHandler ("Refresh_Calendar", RequestCalendarData)
AttachNotificationHandler ("GetCalendarOUI Data", ProcessCalendarData)
AttachPostProxyExecute ("GetCalendarOUI Data", SetDefaultFocus)
```

For more information, see ["AttachEventHandler Method" on page 427](#).

- 5 The presentation model sends the RequestCalendarData method to the Activity HI Calendar Applet proxy.
- 6 The Activity HI Calendar Applet proxy sends a request to the Siebel Server to call the RequestCalendarData method.
- 7 The Siebel Server gets metadata from the Activity HI Calendar Applet that resides on this server, and then sends the GetCalendarOUIData notification method to the presentation model. For more information, see ["About Objects and Metadata" on page 33](#).
- 8 The presentation model does the following:
 - a Runs the ProcessCalendarData method and the SetDefaultFocus method.
 - b Sends the RefreshUI method to the physical renderer. This method gets the relevant properties from the presentation model.
- 9 The physical renderer refreshes the calendar.

4

Example of Customizing Siebel Open UI

This chapter includes a detailed example that describes the typical tasks that you can do to customize Siebel Open UI. It includes the following topics:

- [Roadmap for Customizing Siebel Open UI on page 61](#)
- [Process of Customizing the Presentation Model on page 62](#)
- [Process of Customizing the Physical Renderer on page 84](#)
- [Configuring the Manifest for the Recycle Bin Example on page 94](#)
- [Testing Your Modifications on page 95](#)

Roadmap for Customizing Siebel Open UI

You do the following tasks to customize Siebel Open UI:

- 1 [Process of Customizing the Presentation Model on page 62](#)
- 2 [Process of Customizing the Physical Renderer on page 84](#)
- 3 [Configuring the Manifest for the Recycle Bin Example on page 94](#)
- 4 [Testing Your Modifications on page 95](#)

You can use this sequence as a general guideline to create your own customizations. To summarize, you do the following work:

- **Modify a presentation model.** You customize the presentation model that implements a recycle bin that contains the records that a user deletes in a view. You add a Select list column and modify the Delete button so that the user can choose more than one record, and then delete them from the server database. You configure Siebel Open UI to do a local backup on the client of the chosen records. This configuration requires you to modify the metadata that Siebel Open UI uses in the client and to modify client behavior. It does not require you to modify rendering. So, you only modify the presentation model. You do not modify the physical renderer to implement this part of the example.
- **Modify a physical renderer.** You customize a physical renderer for a third-party carousel control that displays the recycle bin contents and allows the user to restore deleted records. You modify the physical renderer so that Siebel Open UI displays a local back up copy of the deleted records in a carousel control, and then allows the user to choose and restore each of these records. This configuration modifies the physical representation of the records so that Siebel Open UI displays them in a modified grid. It also modifies the physical interactivity that allows the user to choose records in the carousel.

For background information about the architecture that this example uses, see [“Stack That Siebel Open UI Uses to Render Objects” on page 46](#) and [“Life Cycle of User Interface Elements” on page 54](#).

Process of Customizing the Presentation Model

This task is a step in [“Roadmap for Customizing Siebel Open UI” on page 61](#).

To customize the presentation model, do the following tasks:

- 1 [Creating the Presentation Model on page 62](#)
- 2 [Customizing the Setup Logic of the Presentation Model on page 64](#)
- 3 [Customizing the Presentation Model to Identify the Records to Delete on page 66](#)
- 4 [Customizing the Presentation Model to Delete Records on page 69](#)
- 5 [Overriding Predefined Methods in Presentation Models on page 74](#)
- 6 [Customizing the Presentation Model to Handle Notifications on page 75](#)
- 7 [Attaching an Event Handler to a Presentation Model on page 78](#)
- 8 [Customizing Methods in the Presentation Model to Store Field Values on page 81](#)
- 9 [Customizing the Presentation Model to Call the Siebel Server and Delete a Record on page 83](#)

Creating the Presentation Model

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

The presentation model uses the Init method to configure the properties, methods, and bindings of the presentation model, and the Setup method to extract the values that a property set contains. For more information about these methods, see [“Life Cycle of User Interface Elements” on page 54](#).

Figure 17 illustrates the code you use to create the presentation model. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```

if( typeof( SiebelAppFacade.RecycleBinPModel ) == "undefined" ){
    SiebelJS.Namespace( "SiebelAppFacade.RecycleBinPModel" );
    define( "siebel/custom/recyclebinmodel", [], function(){
        SiebelAppFacade.RecycleBinPModel = { function(){
            var consts = SiebelJS.Dependency( "SiebelApp.Constants" );
            function RecycleBinPModel(){
                SiebelAppFacade.RecycleBinPModel.superclass.constructor.apply( this, arguments );
            }
            SiebelJS.Extend( RecycleBinPModel, SiebelAppFacade.ListPresentationModel );
        }
    }
    return RecycleBinPModel;
} ());
return "SiebelAppFacade.RecycleBinPModel";
});

```

The code is annotated with numbered callouts: 2 points to the first line, 3 to the second, 4 to the third, 5 to the fourth, 6 to the fifth, 7 to the sixth, 8 to the seventh, and 9 to the final return statement.

Figure 17. Setting Up the Presentation Model

To create the presentation model

- 1 Create the custom presentation model file:

- a Download a copy of the recyclebinmodel.js file to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

This topic describes how to modify code that resides in the recyclebinmodel.js file. It is recommended that you get a copy of this file to assist in your understanding of how to implement the example that this topic describes. This file includes all the code that this example uses. It also includes more comments that describe code functionality. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

For more information about the folders you can use to store your customizations, see ["Organizing Files That You Customize" on page 122](#).

- b Use a JavaScript editor to open the recyclebinmodel.js file that you downloaded in [Step a](#).

- 2 Verify that the RecycleBinPModel class does not exist and that you do not configure Siebel Open UI to override this class. You add the following code:

```
if(typeof(SiebelAppFacade.RecycleBinPModel) === "undefined"){
```

- 3 Make sure that a namespace exists that Siebel Open UI can use to prevent conflicts:

```
SiebelJS.Namespace("SiebelAppFacade.RecycleBinPModel");
```

- 4 Use the Define method to identify the presentation model file:

```
define("siebel/custom/recyclebinmodel", [], function(){
```

You must use the Define method to make Siebel Open UI can identify the constructor. You must include the relative path and the name of the presentation model file without the file name extension. For more information, see ["Define Method" on page 506](#).

- 5 Define the class:

```
SiebelAppFacade.RecycleBinPModel = (function(){
```

- 6 Load the SiebelApp.Constants namespace that defines the constants that Siebel Open UI uses:

```
var consts = SiebelJS.Dependency("SiebelApp.Constants");
```

- 7 Define the class constructor:

```
function RecycleBinPModel(){
    SiebelAppFacade.RecycleBinPModel.superclass.constructor.apply(this,
    arguments);
}
```

- 8 Set up the injected dependency chain:

```
SiebelJS.Extend(RecycleBinPModel, SiebelAppFacade.ListPresentationModel);
```

For more information about injected dependency chains, see ["About Dependency Injection" on page 69](#).

- 9 Return the constructor:

```

        return RecycleBinPMModel ;
    } ();
    return "Siebel AppFacade. RecycleBinPMModel ";
});

```

10 Save the recyclebinmodel.js file.

Customizing the Setup Logic of the Presentation Model

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

In this topic, you customize the setup logic of the presentation model so that it adds the Selected list column to an applet. You add the control that you configure for this example to the ListColumns list that resides in the client.

Figure 18 illustrates the code you use to customize the setup logic of the presentation model. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```

RecycleBinPMModel.prototype.Setup = function(propSet){
    var mycontrol = SiebelApp.S.App.GetAppletControlInstance (
        "Client Select",
        consts.get("SWE_CTRL_CHECKBOX"),
        "Select",
        "Select",
        "50" );
    this.Get("GetListOfColumns")["SelectionBox"] = mycontrol;
    SiebelAppFacade.RecycleBinPMModel.superclass.Setup.call( this, propSet );
};

```

Figure 18. Customizing the Setup Logic of the Presentation Model

To customize the setup logic of the presentation model

- 1 In the recyclebinmodel.js file, identify the property or method of the object that you must modify.

To do this identification, you can examine methods in the JavaScript API that most closely match the behavior that your example requires. For more information about this JavaScript API, see [Appendix A, “Siebel Open UI Application Programming Interface.”](#)

You can use the following list as a guide to get you started, depending on the area of the Siebel application that your customization must modify:

- **Application methods.** For more information, see [“Application Model Class” on page 478](#).
- **Applet methods.** For more information, see [“Presentation Model Class for Applets” on page 433](#).
- **List applet methods.** For more information, see [“Presentation Model Class for List Applets” on page 452](#).
- **Applet control methods.** For more information, see [“Applet Control Class” on page 466](#).
- **Menu methods.** For more information, see [“Presentation Model Class for Menus” on page 458](#).

- **Siebel business service methods.** For more information, see [“Business Service Class” on page 477](#).

In this example, you can examine the presentation model that Siebel Open UI uses for list applets to identify the property or method of the object that you must modify. To identify this property, see [“Properties of the Presentation Model That Siebel Open UI Uses for List Applets” on page 453](#).

After examining these properties, you find that Siebel Open UI uses the `GetListOfColumns` method that resides in the presentation model. In general, when you examine a property or method in a list applet, it is recommended that you first examine the list presentation model that a list uses, and then the applet presentation model that a form applet uses.

You must add the Selected list column to a list applet. The Selected list column is a control that Siebel Open UI displays in the client. So, you add it to the list of `listOfColumns` that Siebel Open UI already uses.

- 2 Specify the method that the presentation model runs as part of the Setup life cycle:

```
RecycleBINModel.prototype.Setup = function(propSet){
```

In this example, you configure Siebel Open UI to create a control that it displays only in the client, and then insert it into the `GetListOfColumns` property of the applet. You add this code in the Setup life cycle method of the presentation model because this logic is related to the work that Siebel Open UI does to create the applet. Siebel Open UI must create the applet first, and then insert the control. For more information, see [“Summary of Presentation Model Methods” on page 54](#).

- 3 Create a new instance of the AppletControl object:

```
var mycontrol = SiebelApp.S_App.GetAppletControlInstance
```

This example requires Siebel Open UI to create a new `listOfColumns` and add it to the `GetListOfColumns` array. You can use the `GetAppletControlInstance` method to create a new instance of the AppletControl object. For more information, see [“GetAppletControlInstance Method” on page 481](#).

- 4 Name the instance:

```
"Client_Select",
```

You must specify a unique name for the instance. This example uses `Client_Select`, which is a unique value that Siebel Open UI can use to determine the operation that it must perform.

- 5 Specify the control type:

```
const.get("SWE_CTRL_CHECKBOX"),
"Select",
"Select",
"50");
this.Get("GetListOfColumns")["SelectionBox"] = mycontrol;
SiebelAppFacade.RecycleBINModel.superclass.Setup.call(this, propSet);
};
```

where:

- `const.get("SWE_CTRL_CHECKBOX")` specifies the control as a checkbox.

- Select specifies the display name. You can specify any display name.
- 50 specifies the width of the column.

For more information about control types, see [“Applet Control Class” on page 466](#).

- 6 Save the recyclebinmodel.js file.

Customizing the Presentation Model to Identify the Records to Delete

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

In this topic, you modify the list column control that you created in [Step 3 on page 65](#). This control uses a check box, so you must make sure that Siebel Open UI stores the value of this check box if the user toggles it.

Figure 19 illustrates the code that you use to customize the presentation model logic to identify the records to delete. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
RecycleBinPModel.prototype.Init = function(){
    SiebelAppFacade.RecycleBinPModel.superclass.Init.call( this );
    this.AddMethod( "LeaveField", PreLeaveField, { sequence : true, scope : this } );
    this.AddProperty( "DeletionPendingSet", [] );
    .
    .
    .
    function PreLeaveField( control, value, notLeave, returnStructure ){
        if (control.GetName() === "Client Select"){
            this.ExecuteMethod( "SetActiveControl", null );
            var delObj = this.Get( "DeletionPendingSet" );
            var currentSelection = this.Get( "GetSelection" );
            if( value === "Y" ){
                delObj[currentSelection] = this.Get( "GetRecordSet" )[currentSelection];
            }
            else{
                delObj[currentSelection] = null;
            }
            returnStructure[ "CancelOperation" ] = true;
            returnStructure[ "ReturnValue" ] = true;
        }
    }
}
```

Figure 19. Customizing the Presentation Model Logic to Identify the Records to Delete

To customize the presentation model to identify the records to delete

- 1 In the recyclebinmodel.js file, add the method that Siebel Open UI must call:


```
this.AddMethod("LeaveField", PreLeaveField, {sequence: true, scope: this});
```

 where:

- AddMethod adds the LeaveField method.

To identify the method to add when you do your own customization work, you can examine the flowcharts for the life cycle that Siebel Open UI uses that meets your business requirement. To view these flowcharts, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

In this example, the business requirement is to save the value in a control. Siebel Open UI saves the value of a control when the user navigates away from the control, so it calls the LeaveField method to handle this requirement. For more information, see [“LeaveField Method” on page 444](#) and [“Flow That Handles Focus Changes in List Applets” on page 529](#).

- PreLeaveField, {sequence : true, scope : this} configures Siebel Open UI to call your custom LeaveField method before it calls the predefined LeaveField method. It does this during the Init life cycle when it runs the AddMethod method. It is recommended that you set up the presentation model methods at the beginning of the Init life cycle call that contains most of the properties and dependency injections, including predefined and custom methods. For more information about Init, see [“Life Cycle of User Interface Elements” on page 54](#). For more information, see [“About Dependency Injection” on page 69](#).

It is recommended that you use a named method to specify the Prexxx customization method, such as PreLeaveField. This configuration makes sure that Siebel Open UI uses the same method for all presentation model instances. It is not recommended that you specify the Prexxx customization method as an anonymous method in the AddMethod call because Siebel Open UI creates this anonymous method for every instance of the presentation model that resides in memory, possibly for more than one applet in the same view. Defining an anonymous method in this situation might cause a conflict.

- 2 Create the condition:

```
if (ctrl.GetName() === "Client_Select"){
```

The Setup method uses the GetName method with a literal return value of Client_Select. It identifies the method that Siebel Open UI uses for your custom control. For more information, see [“GetName Method for Applet Controls” on page 470](#).

- 3 Make sure Siebel Open UI returns your custom logic after it sets the CancelOperation part of the return value to true:

```
returnStructure[ "Cancel Operation" ] = true;
```

This configuration overrides the predefined code when Siebel Open UI calls LeaveField for your new list column. In this example, you must implement LeaveField for the control, so it is not desirable to call the predefined code for this control after Siebel Open UI finishes running your customization of the LeaveField method. For more information about using ReturnStructure when you modify a method, see [“AddMethod Method” on page 425](#).

- 4 Configure Siebel Open UI to return a value of true after it sets the CancelOperation part of returnStructure to true:

```
returnStructure[ "ReturnValue" ] = true;
```

The LeaveField method returns a value of true to indicate success in this example, so you must make sure Siebel Open UI uses the same logic after your customization finishes running and returns a value. This configuration makes sure the Init life cycle continues on the success path after the custom LeaveField method runs. You can use ReturnValue to make sure Siebel Open UI sets the return value of your custom implementation to the required value. In this example, you set this value to true.

- 5 Disable the processing that Siebel Open UI does for the control that is in focus:

```
this.s.ExecuteMethod("SetActiveControl", null);
```

This code sets the active control to null. For more information, see [“Disabling Automatic Updates” on page 69](#) and [“SetActiveControl Method” on page 447](#).

- 6 Add the property that Siebel Open UI uses to store the set of records that are pending deletion:

```
this.s.AddProperty("DeletionPendingSet", []);
```

The set of records that are pending deletion represent the state of your custom presentation model, so you add the DeletionPendingSet property to store the field values for this set of records.

- 7 Identify the records that Siebel Open UI must delete:

```
var delObj = this.s.Get("DeletionPendingSet");
var currentSelection = this.s.Get("GetSelection");
if(value === "Y"){
    delObj[currentSelection] = this.s.Get("GetRecordSet")[currentSelection];
}
else{
    delObj[currentSelection] = null;
}
```

Siebel Open UI must identify the records that the user chooses to delete so that it can populate a value into the DeletionPendingSet property.

To identify this property, you can examine the properties that the presentation model uses for the applet. This work is similar to the work you do in [Step 1 on page 64](#) to identify the property in the presentation model that Siebel Open UI uses for lists, except in this topic you examine the properties described in [“Properties of the Presentation Model That Siebel Open UI Uses for Applets” on page 436](#).

After examining these properties, you find that Siebel Open UI uses the GetSelection property to get the index of the record that the user has chosen from among all the records that Siebel Open UI displays. You also find that you can use the GetRecordSet property to get this full set of records.

- 8 Save the recyclebinmodel.js file.

About Dependency Injection

Dependency injection is a software technique that Siebel Open UI uses to create a dependency between a presentation model and a physical renderer. If Siebel Open UI modifies a method or property that resides in the presentation model, then it also modifies a method or property that resides in the physical renderer. It allows Siebel Open UI to implement logic at run time rather than during a compile. These dependency injections allow Siebel Open UI to use an *injected dependency chain*, which is a series of two or more dependency injections. You can modify Siebel Open UI to make this chaining depend on conditions that Siebel Open UI modifies at run time. It can use all the methods that the Init method references in [“Summary of Presentation Model Methods” on page 54](#) for dependency injection. For an example that uses dependency injection, see [“Customizing the Physical Renderer to Refresh the Recycle Bin” on page 89](#).

Disabling Automatic Updates

Siebel Open UI sends updated field values to the Siebel Server for any fields that the user has modified in the client. In this example, you must disable this update functionality for the current control. You can reference the documentation for the predefined applet to identify the presentation model property that you must modify. In this situation, the documentation indicates that you can configure Siebel Open UI to use the SetActiveControl property of the active control on the applet and set it to null. For more information, see [“Disabling Automatic Updates” on page 69](#), [“SetProperty Method” on page 432](#), and [“SetActiveControl Method” on page 447](#).

ExecuteMethod calls a method that resides in the presentation model. It makes sure that Siebel Open UI runs all injected dependency chains that the method requires when it runs. You must use ExecuteMethod to call any predefined or custom method that resides in a presentation model. For more information, see [“About Dependency Injection” on page 69](#) and [“ExecuteMethod Method” on page 430](#).

Customizing the Presentation Model to Delete Records

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

Figure 20 illustrates the code you use to configure the presentation model to delete records. In this topic, you configure Siebel Open UI to customize and conditionally override the InvokeMethod method. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
RecycleBinPMModel.prototype.Init = function(){
    SiebelAppFacade.RecycleBinPMModel.superclass.Init.call( this );
    this.AddMethod( "InvokeMethod", PreInvokeMethod, { sequence : true, scope : this } );
    this.AddProperty( "ObjectsUnderDeletion", [] );
    .
    .
    .
}

function PreInvokeMethod( methodName, psInputArgs, lp, returnStructure ){
    if( methodName === "DeleteRecord" && !this.Get( "InDeletion" ) ){
        this.SetProperty( "InDeletion", true );
        var deletionPending = this.Get( "DeletionPendingSet" );
        if( deletionPending.length > 0 ){
            for( var counter = deletionPending.length - 1; counter >= 0; counter-- ){
                var currentObj = deletionPending[counter];
                if( currentObj ){
                    this.ExecuteMethod( "SetActiveControl", null );
                    this.ExecuteMethod( "HandleRowSelect", counter, false, false );
                    if( this.ExecuteMethod( "CanInvokeMethod", "DeleteRecord" ) ){
                        this.Get( "ObjectsUnderDeletion" )[ this.Get( "GetSelection" ) ] = currentObj;
                        var inputPS = SiebelApp.S_App.NewPropertySet();
                        this.ExecuteMethod( "InvokeMethod", "DeleteRecord", inputPS );
                    }
                }
            }
        }
        this.SetProperty( "DeletionPendingSet", [] );
        returnStructure [ "CancelOperation" ] = true;
        SiebelApp.S_App.uiStatus.Free();
        this.SetProperty( "InDeletion", false );
    }
}
```

Figure 20. Customizing the Presentation Model to Delete Records

To customize the presentation model to delete records

- 1 In the recyclebinpmmodel.js file, add the method that Siebel Open UI uses to delete a record:

```
this.AddMethod( "InvokeMethod", PreInvokeMethod, {sequence: true, scope: this} );
```

You must identify the method that Siebel Open UI uses when the user clicks Delete. To do this identification, it is recommended that you examine the flowchart that Siebel Open UI uses during a typical life cycle when it calls methods that reside on the Siebel Server. The life cycle flowchart indicates that Siebel Open UI calls the DeleteRecord method when it calls the InvokeMethod method. You add this code in the Init method. For more information, see [“Life Cycle Flows That Create New Records in List Applets” on page 531](#) and [“DeleteRecord Method” on page 395](#).

This configuration is similar to the configuration you added in [Step 1 on page 66](#) that includes the AddMethod method and the sequence statement.

- 2 Call the custom logic only if Siebel Open UI calls the DeleteRecord method:

```
if ( (methodName === "DeleteRecord") && !this.Get( "InDeletion" ) ){
```

This code examines the value of the InDeletion property that you set up in [Step 3](#).

- 3 Set the InDeletion property to true only if Siebel Open UI starts the deletion process:

```
this.s.SetProperty("InDeletion", true);
```

This code determines whether or not Siebel Open UI is already running an instance of your custom delete process, and then makes sure that no more than one of these instances runs at the same time. The InDeletion property determines whether or not the deletion process is currently running.

You could use the following code in the Init method to add this property:

```
this.s.AddProperty("InDeletion", false)
```

This example demonstrates how you can use SetProperty to use a property temporarily so that it is similar to a conditional flag. This example uses SetProperty to create this property only when necessary. If Siebel Open UI calls the Get method before it calls the SetProperty method, then the JavaScript returns a value of undefined, which is the default value that JavaScript assigns to any variable that is not defined.

- 4 Get the set of records where the Selected value of each of these records includes a check mark:

```
var deletionPending = this.s.Get("DeletionPendingSet");
```

This code gets the state of the set of records before the user clicks Delete. Siebel Open UI stores this information in the DeletionPendingSet property in the LeaveField customization that you added in [Step 6 on page 68](#).

- 5 Determine whether or not the user has chosen at least one record for deletion:

```
if (deletionPending.length > 0){
```

This code represents this condition as > 0 , where 0 indicates the number of records chosen.

- 6 Iterate through all the records that the user has chosen to delete:

```
for (var counter = deletionPending.length - 1; counter >= 0; counter--){
    var currentObj = deletionPending[counter];
    if (currentObj){
    }
}
```

- 7 Disable the processing that Siebel Open UI does for the control that is in focus:

```
this.s.ExecuteMethod("SetActiveControl", null);
```

For more information, see ["Disabling Automatic Updates" on page 69](#) and ["SetActiveControl Method" on page 447](#).

- 8 Modify the application state so that Siebel Open UI references the record that it must delete:

```
this.ExecuteMethod("HandleRowSelect", counter, false, false);
```

To identify this code when you customize Siebel Open UI, it is recommended that you examine [“Flow That Handles Navigation to Another Row in List Applets” on page 536](#). In this example, this flow indicates that you must use the HandleRowSelect method. HandleRowSelect resides in the presentation model that Siebel Open UI uses for list applets, so you can configure Siebel Open UI to use ExecuteMethod to call it. For more information, see [“HandleRowSelect Method” on page 455](#).

- 9 Make sure that Siebel Open UI can call the DeleteRecord method:

```
if(this.ExecuteMethod("CanInvokeMethod", "DeleteRecord")){
```

It is recommended that you configure Siebel Open UI to call CanInvoke before it calls a method to make sure that it can call this method in the context of the object that is currently in scope. Siebel Open UI can use the CanInvoke method to model complex logic for any record that exists in the Siebel Database on the Siebel Server. This logic can determine whether or not Siebel Open UI can call an operation according to the scope that it applies to a current object, such as a record that is in scope. In this example, it determines whether or not it can call the DeleteRecord method.

You can use the method descriptions in [Appendix A, “Siebel Open UI Application Programming Interface”](#) to identify the method that you must use in your customization work.

For more information about the method that this example uses, see [“CanInvokeMethod Method for Presentation Models” on page 437](#).

- 10 Add a property that Siebel Open UI can use to store information about the records that it sends to the Siebel Server for deletion:

```
this.AddProperty("ObjectsUnderDeletion", []);
```

Delete confirmation occurs through an asynchronous notification, so Siebel Open UI must back up the information that describes the record that it is about to delete. The notification handler requires this information so that it can do the post-processing work for this record. Subsequent steps in this topic describe this notification handler. For more information, see [“About Synchronous and Asynchronous Requests” on page 74](#) and [“Notifications That Siebel Open UI Supports” on page 541](#).

- 11 Delete the record:

```
this.Get("ObjectsUnderDeletion")[this.Get("GetSelection")] = currentObj;
var inputPS = SiebelApp.S_App.NewPropertySet();
this.ExecuteMethod("InvokeMethod", "DeleteRecord", inputPS);
```

where:

- ObjectsUnderDeletion inserts the record into a backed up record set, and if this insert occurs at an index location that is equal to the index of the selected row, then Siebel Open UI can reference the selected row to identify the correct index to use when processing the NotifyDeleteNotification reply. The Siebel Server sends this reply. Siebel Open UI must identify the record where it set the notification when it handles the NotifyDeleteNotifications notification. You can configure Siebel Open UI to call HandleRowSelect to select the row before it sends the request to delete the record.

- `GetSelection` is a property of the applet presentation model that includes an index that identifies the chosen record. This record resides in the record set that resides in the client. When you develop your own customization, you can reference the documentation to identify the property that your customization requires. For more information, see ["Properties of the Presentation Model That Siebel Open UI Uses for Applets" on page 436](#).
- `InvokeMethod` is a method that resides in the presentation model that Siebel Open UI uses for a list applet. You can use `ExecuteMethod` to call it.
- `false` configures Siebel Open UI to make a synchronous request to the Siebel Server. A synchronous request makes sure that Siebel Open UI sends all `DeleteRecord` requests to the server before it exits the loop. If Siebel Open UI exits the loop during a synchronous request, then it sends all `DeleteRecord` requests sequentially. In this situation, Siebel Open UI sends the requests to the server so that the server can process a reply for the previous request, including the delete completion notifications. The server does this processing during a synchronous request before it sends the next `DeleteRecord` request. You can also use an asynchronous request where Siebel Open UI sends all the `DeleteRecord` requests to the server before it processes any of the replies for these requests. For more information, see ["About Synchronous and Asynchronous Requests" on page 74](#).

12 Set the `DeletionPendingSet` property to zero:

```
this.s.SetProperty("DeletionPendingSet", 0);
```

This code sets the `DeletionPendingSet` property to zero after Siebel Open UI finishes all the `DeleteRecord` calls on the Siebel Server.

13 Set the `CancelOperation` member of the `returnStructure` to true:

```
returnStructure["CancelOperation"] = true;
```

You configure Siebel Open UI to set this member before it exits the outer loop that processes the `deletionPending` records. You do this so that Siebel Open UI does not use the `DeleteRecord` argument to make another call to the predefined `InvokeMethod` method. For more information about `ReturnStructure`, see ["AddMethod Method" on page 425](#).

14 Set the `InDeletion` flag to false:

```
this.s.SetProperty("InDeletion", false);
```

You configure Siebel Open UI to set this property before it exits the conditional block that does the `InvokeMethod` processing for the `DeleteRecord` method.

15 Save the `recyclebinmodel.js` file.

About Synchronous and Asynchronous Requests

A *synchronous request* is a type of request that Siebel Open UI sends to the Siebel Server, and then waits for a reply to this request before it continues any other processing. An *asynchronous request* is a type of request that Siebel Open UI sends to the Siebel Server, and then continues other processing while it waits for a reply to this request. The GetSelection request is synchronous, so Siebel Open UI cannot send another request to move the selection to a different record before the Siebel Server sends a reply notification that indicates a successful deletion. When processing this notification, the intended row remains in the same row that Siebel Open UI most recently selected. Siebel Open UI can use the selected row as a common index that it can use to reference the record. For information about how you can configure an asynchronous request, see [“Allowing Users to Interact with Clients During Business Service Calls” on page 120](#).

Overriding Predefined Methods in Presentation Models

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

If Siebel Open UI calls the GetFormattedFieldValue method for a control that it only displays in the Siebel Open UI client, then this client cannot find the field in the list of fields that it uses, and the client creates an error. To avoid this situation, you can customize Siebel Open UI to override the predefined GetFormattedFieldValue method so that it does not create an error when it calls GetFormattedValue for your new list column.

To override predefined methods in presentation models

- 1 Use the flowcharts to identify the method that you must modify.

Siebel Open UI displays values for applet controls and list columns after it gets these values from the client. It caches these values in the client after it downloads them from the Siebel Server. To identify the method that handles these values, you can examine the flowchart that describes how Siebel Open UI creates a new record in a list applet, and then updates the client. In this example, the flowchart indicates that it calls the GetFormattedFieldValue method. If the physical renderer requires the ShowControlValue method, then it calls the presentation model to run the GetFormattedFieldValue method. For more information, see [“Flow That Creates New Records in List Applets, Updating the User Interface” on page 534](#).

- 2 In the recyclebinmodel.js file, configure Siebel Open UI to conditionally override and customize the method:

```
RecycleBinPMModel.prototype.Init = function(){
    SiebelAppFacade.RecycleBinPMModel.superclass.Init.call(this);
    this.AddMethod("GetFormattedFieldValue", PreGetFormattedFieldValue,
    {sequence: true, scope: this});
    .
    .
    .
    function PreGetFormattedFieldValue(control, bUseWS, recIndex, returnStructure){
        if (control.GetName() === "Client_Select"){
            returnStructure["Cancel Operation"] = true;
        }
    }
}
```

```
        returnStructure["ReturnVal ue"] = "";
    }
}
```

where:

- `this.AddMethod` adds the `PreGetFormattedFieldValue` method in the `Init` life cycle and specifies `PreGetFormattedFieldValue` as the customization.
- `sequence : true` specifies to call the custom `PreGetFormattedFieldValue` before it calls the predefined `GetFormattedFieldValue` method.
- The following code in the custom method determines whether or not the control that Siebel Open UI is currently examining is the client-only control:

```
if (control.GetName() === "Client_Select")
```

If it is, then Siebel Open UI sets the `CancelOperation` member of the `returnStructure` to `true` and the `ReturnValue` to `null`. For more information about `returnStructure`, see [“AddMethod Method” on page 425](#).

- 3 Save the `recyclebinmodel.js` file.

Customizing the Presentation Model to Handle Notifications

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

The Siebel Server sends a confirmation of the record deletion when it receives the `InvokeMethod` request for the `DeleteRecord` method. You can write a handler for the `NotifyDeleteRecord` notification to process this confirmation in the client. For more information, see [“DeleteRecord Method” on page 395](#).

Siebel Open UI packages a notification that it gets from the Siebel Server in the business component layer as part of a reply property set. This property set includes information about server state modifications or replies to requests for state information. For example, if Siebel Open UI deletes a record on the server, then it sends a `NotifyDeleteRecord` notification to the client. When Siebel Open UI sends a request to the server, the server processes the request, Siebel Open UI examines the relevant modifications that exist on the server, and then collects and packages notifications that are ready to communicate to the client. If Siebel Open UI sends an `InvokeMethod` call for the `DeleteRecord` method to the server, then the Siebel Web Engine sends a `NotifyDeleteRecord` notification from the business component layer to the client. For more information about the business component layer, see *Configuring Siebel Business Applications*.

Figure 21 illustrates the code you use to customize the presentation model to handle notifications. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```

RecycleBinPModel.prototype.Init = function(){
  SiebelAppFacade.RecycleBinPModel.superclass.Init.call( this );
  this.AttachNotificationHandler( consts.get( "SWE_PROP_BC_NOTI_DELETE_RECORD" ), HandleDeleteNotification );
  this.AddMethod( "RefreshList", function(){} );
  this.AddProperty( "DeletionCompleteSet", [] );
  .
  .
  .
}

function HandleDeleteNotification(propSet){
  var objectsUnderDeletion = this.Get( "ObjectsUnderDeletion" );
  if( objectsUnderDeletion.length > 0 ){
    var activeRow = propSet.GetProperty( consts.get( "SWE_PROP_BC_NOTI_ACTIVE_ROW" ) );
    if( activeRow == this.Get( "GetSelection" ) && objectsUnderDeletion[ activeRow ] ){
      this.Get( "DeletionCompleteSet" ).push( objectsUnderDeletion[ activeRow ] );
      objectsUnderDeletion[ activeRow ] = null;
      this.ExecuteMethod( "RefreshList" );
    }
  }
}

```

Figure 21. Customizing the Presentation Model to Handle Notifications

To customize the presentation model to handle notifications

- 1 Identify the notification type that Siebel Open UI must handle.

Examine the notification types in the [“Notifications That Siebel Open UI Supports”](#) on page 541 topic. Look for a notification type that indicates it might include the information that your customization requires. For this example, the notification type for the NotifyDeleteRecord notification is SWE_PROP_BC_NOTI_DELETE_RECORD.

- 2 Examine the methods in the presentation model that indicate they might be useful for your customization.

The AttachNotificationHandler method is the appropriate method to use for this example. For more information, see [“AttachNotificationHandler Method”](#) on page 428.

- 3 In the recyclebinpmodel.js file, add the AttachNotificationHandler to the Init method of the presentation model:

```

this.AttachNotificationHandler( consts.get( "SWE_PROP_BC_NOTI_DELETE_RECORD" ),
  HandleDeleteNotification );

```

- 4 Add the custom method that Siebel Open UI uses to handle replies from NotifyDeleteRecord and to populate the recycle bin:

```

function HandleDeleteNotification(propSet){

```

- 5 Get the property that you use to identify the objects that are currently flagged for deletion:

```

var objectsUnderDeletion = this.Get( "ObjectsUnderDeletion" );

```

You configured this property in [Step 10 on page 72](#) to back up the records that Siebel Open UI is in the process of deleting.

- 6 Determine whether or not any records exist in the In Progress list:

```
if(objectsUnderDeletion.Length > 0){
```

Siebel Open UI must process these records and move them to the recycle bin. In this step and in several subsequent steps, you do more than one examination to make sure the notification instance that Siebel Open UI is handling is the instance that it requires for the notification handler. Some repeating notifications might exist that you must process to avoid duplication.

7 Identify the row involved with the NotifyDeleteRecord notification:

```
var activeRow = propSet.GetProperty(consts.get("SWE_PROP_BC_NOTI_ACTIVE_ROW"));
```

In this example, you use the SWE_PROP_BC_NOTI_ACTIVE_ROW property. For more information about this property, see ["Summary of Notifications That Siebel Open UI Supports" on page 542](#).

8 Make sure that this notification confirms the deletion, and make sure that this notification is not a duplicate:

```
if(activeRow == this.Get("GetSelection") && objectsUnderDeletion[activeRow]){
```

where:

- The following code determines if the record that the NotifyDeleteRecord method references is the currently selected record:

```
activeRow == this.Get("GetSelection")
```

This example uses a synchronous request, so Siebel Open UI selects the record that the DeleteRecord method references in the context of PreInvokeMethod. It selects no other record after it makes this initial selection while the Siebel Server sends the delete confirmation notification to the client. For more information, see ["About Synchronous and Asynchronous Requests" on page 74](#).

- The following code makes sure that this notification is not a duplicate:

```
objectsUnderDeletion[ activeRow ]
```

It determines whether or not Siebel Open UI has already removed the record that it is examining in a previous instance of handling the same notification for the same record.

9 Add a property that Siebel Open UI can use to store the list of records that the user deletes but might retrieve from the recycle bin:

```
this.AddProperty("DeletionCompletedSet", []);
```

10 Store the deleted record:

```
this.Get("DeletionCompletedSet").push(objectsUnderDeletion[activeRow]);
```

The conditional block where this code resides determines that this notification is not a duplicate NotifyDeleteRecord notification for the record that the DeleteRecord method requests deletion. So, this push statement pushes the deleted record into the DeletionCompletedSet property that you defined [Step 9](#).

11 Remove the record from the Deletion in Progress list:

```
objectsUnderDeletion[ activeRow ] = null;
```

12 Add the RefreshList method:

```
this.s.AddMethod("RefreshList", function(){});
```

Siebel Open UI must refresh the recycle bin after [Step 11](#) adds a record to this recycle bin. You can use dependency injection through the AttachPMBinding method to inform the physical renderer that the recycle bin requires a refresh. For more information, see ["About Dependency Injection" on page 69](#). For more information, see ["How Siebel Open UI Uses Nondetailed Data to Indicate Modifications That Occur in Detailed Data" on page 78](#).

13 Run the RefreshList method:

```
this.s.ExecuteMethod("RefreshList");
```

14 Save the recyclebinmodel.js file.

How Siebel Open UI Uses Nondetailed Data to Indicate Modifications That Occur in Detailed Data

Siebel Open UI uses the dependency that exists between the presentation model and the physical renderer to indicate a high-level modification in a property or method, such as a modifying in the list of records that it must display. This dependency configures Siebel Open UI to run a high-level renderer method, such as a method that repopulates the entire physical markup of columns and data in the grid container. The renderer method then gets the detailed presentation model attributes, such as columns and data, through properties or methods that the presentation model contains.

This example uses the RefreshList method as an indicator that Siebel Open UI modified something in the DeletionCompletedSet property. When you configure the physical renderer in ["Customizing the Physical Renderer to Refresh the Recycle Bin" on page 89](#), you configure Siebel Open UI to use AttachPMBinding to bind a physical renderer method to the RefreshList method. You also configure Siebel Open UI to use this physical renderer method to get the detailed data that the DeletionCompletedSet method references. Siebel Open UI gets this data from the presentation model so that the physical renderer can render it. For more information, see ["AttachPMBinding Method" on page 428](#).

Attaching an Event Handler to a Presentation Model

This task is a step in ["Process of Customizing the Presentation Model" on page 62](#).

At this point in this example, you have set up and customized the presentation model to choose records to delete, to delete them, and then to move them to the recycle bin. In this topic, you modify the presentation model to allow the user to click an item in the carousel, and then click the plus sign (+) to restore the record.

Figure 22 illustrates the code you use to attach an event handler to a presentation model. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
RecycleBinPMModel.prototype.Init = function(){
  SiebelAppFacade.RecycleBinPMModel.superclass.Init.call( this );
  this.AttachEventHandler( "RESTORE", OnClickRestore );
  this.AddProperty( "restorationIndex", -1 );
  .
  .
  .
function OnClickRestore(index){
  if( this.ExecuteMethod( "CanInvokeMethod", "NewRecord" ) ){
    this.SetProperty( "inRestoration", true );
    this.SetProperty( "restorationIndex", index );
    this.ExecuteMethod( "InvokeMethod", "NewRecord", null, false );
    this.ExecuteMethod( "InvokeMethod", "WriteRecord", null, false );
  }
}
```

Figure 22. Attaching an Event Handler to a Presentation Model

To attach an event handler to a presentation model

- 1 In the recyclebinpmmodel.js file, add the method that handles the event. Siebel Open UI calls this method when the user clicks the plus sign (+):

```
function OnClickRestore(index){
```

The name of an event handler typically starts with the following prefix:

```
On
```

- 2 Bind the OnClickRestore method to the RESTORE custom event:

```
this.AttachEventHandler("RESTORE", OnClickRestore);
```

This code adds the RESTORE custom event. The physical renderer sends this event to the presentation model, and then the presentation model runs OnClickRestore. The AttachEventHandler method sets up a dependency injection, so you add it in the Init method. For more information, see [“AttachEventHandler Method” on page 427](#) and [“About Dependency Injection” on page 69](#).

- 3 Identify the method that Siebel Open UI uses when a user creates a record.

Examine the [“Flow That Creates New Records in List Applets, Calling the Siebel Server” on page 532](#). Note that Siebel Open UI uses the NewRecord method, and then the WriteRecord method as an input argument for the InvokeMethod method when it runs InvokeMethod in the presentation model. For more information, see [“NewRecord Method” on page 474](#).

- 4 Determine how Siebel Open UI stores the field values of a new record that a user creates.

Examine [“Flow That Handles Focus Changes in List Applets” on page 529](#). This flow describes the process that occurs between the initial NewRecord call and the WriteRecord call when Siebel Open UI creates a record in the client. It stores the field values in the client while the user enters these values and navigates from one field to another field. For more information, see [“WriteRecord Method” on page 407](#).

Siebel Open UI can do the following to create a record that it restores through the OnClickRestore event handler:

- Run the InvokeMethod method for the NewRecord.
- Store values that the user enters in each field, and use values from the records that Siebel Open UI stores in the recycle bin.
- Run the InvokeMethod method for WriteRecord with the client already configured to include the field values for the record.

- 5 Make sure Siebel Open UI can use the NewRecord method in the applet:

```
if (this.s.ExecuteMethod("CanI nvokeMethod", "NewRecord")){
```

If Siebel Open UI cannot run the NewRecord method, then it exits this conditional statement.

- 6 Add the property that Siebel Open UI uses to store the index that identifies the record it must restore:

```
this.s.AddProperty("restorati onI ndex", -1);
```

The physical renderer must specify the record to restore. To do this, it uses the DeletionCompletedSet property to get the restorationIndex of this record from the client and store it. It then sends this index to the presentation model as part of a request to restore the record. The restorationIndex is an index that resides in the DeletionCompletedSet property of the record.

Siebel Open UI sends this value from the recycle bin record that the user chooses to restore. The OnClickRestore method receives this property and Siebel Open UI then stores this value in the restorationIndex property of the presentation model.

- 7 Configure the OnClickRestore method:

```
this.s.SetProperty("i nRestorati on", true);
this.s.SetProperty("restorati onI ndex", i ndex);
this.s.ExecuteMethod("I nvokeMethod", "NewRecord", nul l, fal se);
this.s.ExecuteMethod("I nvokeMethod", "Wri teRecord", nul l, fal se);
```

where:

- NewRecord and WriteRecord are input arguments to the InvokeMethod method. In [Step 3](#) you determined that Siebel Open UI uses the NewRecord method or the WriteRecord method as an input argument for the InvokeMethod, so you specify these methods in this code.

Siebel Open UI stores the field values of a record in the WriteRecord request before it sends this request to the Siebel Server. It stores these values differently depending on whether it creates a record from the recycle bin or whether the user creates a new record. The physical user interface layer does not store these values if the user attempts to restore a record from the recycle bin. It stores these values only if the user creates a new record. You can write the logic that stores these values in a Prexxx customization method of the WriteRecord method. You write this customization in the next topic in this example, [“Customizing Methods in the Presentation Model to Store Field Values” on page 81](#).

This customization runs only while WriteRecord is running to restore a record from the recycle bin. It does not run when the user creates a new record and Siebel Open UI calls WriteRecord. When you start this restoration logic in the OnClickRestore method, you set a presentation model property that serves as a flag that indicates that a recycle bin restoration is in progress. An explicit AddProperty call does not exist for this property, so Siebel Open UI creates this property only if the user uses the recycle bin.

- 8 Save the recyclebinmodel.js file.

Customizing Methods in the Presentation Model to Store Field Values

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

In this topic, you use the ExecuteMethod method to store the values of the record that the user is attempting to restore from the recycle bin.

[Figure 23](#) illustrates the code you use to customize a method in the presentation model to store the field values of records. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
function PreInvokeMethod( methodName, psInputArgs, lp, returnStructure){
    if( methodName === "DeleteRecord" && !this.Get( "InDeletion" ) ){
        .
        .
        .
    }
    else if( methodName === "WriteRecord" && this.Get("inRestoration") ){
        var recordSet = this.Get( "DeletionCompleteSet" );
        var record = recordSet[ this.Get( "restorationIndex" ) ];
        var listOfColumns = this.Get( "ListOfColumns" );
        var controls = this.Get( "GetControls" );
        for( var i = 0, len = listOfColumns.length; i < len; i++ ){
            var control = controls[ listOfColumns[ i ].name ];
            if( control ){
                this.ExecuteMethod( "LeaveField", control, record[control.GetFieldName()], true);
            }
        }
    }
}
```

Figure 23. Customizing a Method in the Presentation Model to Store the Field Values of Records

To customize methods in the presentation model to store field values

- 1 In the recyclebinmodel.js file, add a condition that makes sure Siebel Open UI runs the customization logic only if the user is restoring a record from the recycle bin, and not adding a new record: (

```
else if(methodName === "WriteRecord" && this.Get("inRestoration")){
```

This if statement examines the value of the methodName in the WriteRecord argument and the value of the inRestoration property. For more information, see ["WriteRecord Method" on page 407](#).

- 2 Get the set of records for the recycle bin:

```
var recordSet = this.Get("DeletionCompleteSet");
```

In [Step 10 on page 77](#), you configured the DeletionCompletedSet property of the presentation model to store each record that the user adds to the recycle bin.

- 3 Get the back up copy of the record that the physical renderer requests to restore:

```
var record = recordSet[this.Get("restorationIndex")];
```

To get this value, you access the restorationIndex property that you added in [Step 6 on page 80](#).

- 4 Identify the method that Siebel Open UI uses to indicate that the user navigated away from an applet.

To do this, you can examine ["Flow That Handles Focus Changes in List Applets" on page 529](#). Note that Siebel Open UI calls the LeaveField method as the last step in the flow. The LeaveField method determines whether or not Siebel Open UI removed the focus from a field in an applet, so Siebel Open UI uses this step in the flow as a flag to signal that it must store the field values. To get information about the methods that the flowcharts describe when you develop your own customization, you can use the descriptions in [Appendix A, "Siebel Open UI Application Programming Interface."](#)

- 5 Get the list of columns that the list applet contains. This list is identical to the list of columns that the DeletionCompleteSet property contains:

```
var listOfColumns = this.Get("ListOfColumns");
```

- 6 Get the list of controls that the list applets contains:

```
var controls = this.Get("GetControls");
```

For more information about the GetControls property, see ["Properties of the Presentation Model That Siebel Open UI Uses for Applets" on page 436](#).

- 7 Store the field values:

```
for(var i = 0, len = listOfColumns.length; i < len; i++){
    var control = controls[listOfColumns[i].name];
    if(control){
        this.ExecuteMethod("LeaveField", control, record[control.GetFieldName()],
            true);}
    }
}
```

where:

- The following code iterates through the applet controls that correspond to the list columns of the record that the DeletionCompleteSet property identifies:

```
for(var i = 0, len = listOfColumns.length; i < len; i++)
```

- this.ExecuteMethod calls the LeaveField method that you identified in [Step 4](#). It calls this method one time for each iteration. It sends the field value from the corresponding control of the record that DeletionCompleteSet identifies. It sends this value to an argument. When this code iterates, it runs the LeaveField method for every control that Siebel Open UI must populate in the new record that it is using to restore the deleted record from the recycle bin.

Siebel Open UI must send the LeaveField method as a control and store a value for this control. In this example, Siebel Open UI iterates through each control that the list applet contains and sends the value of the corresponding list column that it uses for the control from the record that the DeletionCompleteSet property gets in [Step 2](#).

For a description of the arguments that LeaveField uses, [“Summary of Methods That You Can Use with the Presentation Model for Applets” on page 435](#).

- record stores the field value of the record that Siebel Open UI is restoring. The subsequent WriteRecord call packages and sends these values to the Siebel Server. Siebel Open UI stores these values when it runs the LeaveField method. For more information about this flow, see [“Flow That Handles Focus Changes in List Applets” on page 529](#).

- 8 Save the recyclebinmodel.js file.

Customizing the Presentation Model to Call the Siebel Server and Delete a Record

This task is a step in [“Process of Customizing the Presentation Model” on page 62](#).

In this topic, you configure the presentation model to remove the record from the recycling bin. You use a dependency injection to call a method on the Siebel Server after the stack that Siebel Open UI uses to call the server has finished processing. For more information, see [“About Dependency Injection” on page 69](#) and [“Customizing Events” on page 118](#).

To customize the presentation model to call the Siebel Server and delete a record

- 1 In the recyclebinmodel.js file, add the following code to the Init method in the presentation model:

```
this.AttachPostProxyExecuteBinding("WriteRecord", PostWriteRecord);
```

You use the Init method to send a WriteRecord call to the Siebel Server. For more information, see [“WriteRecord Method” on page 407](#) and [“AttachPostProxyExecuteBinding Method” on page 429](#).

- 2 Add the following code anywhere in the presentation model:

```
function PostWriteRecord(methodName, inputPS, outputPS){
  if(this.Get("inRestoration")){
    this.Get("DeletionCompleteSet")[this.Get("restorationIndex")] = null;
```

```
        this.s.ExecuteMethod("RefreshList");  
        this.s.SetProperty("inRestoration", false);  
    }
```

where:

PostWriteRecord does the following work:

- Removes the record that Siebel Open UI restored in [Step 7 on page 82](#). It removes this record from the DeletionCompleteSet property.
- Calls the RefreshList method to start another round of binding to the physical renderer. In the next topic in this example you configure Siebel Open UI to call the HandleDeleteNotification method to refresh the list and remove the record from the recycle bin in the client.
- Sets the inRestoration property of the presentation model to false. You set up this property to true in step [Step 7 on page 80](#) to indicate that Siebel Open UI is restoring a record. The restoration is now finished so you can configure Siebel Open UI to set inRestoration to false.

- 3 Save the recyclebinmodel.js file.

Process of Customizing the Physical Renderer

This task is a step in ["Roadmap for Customizing Siebel Open UI" on page 61](#).

To customize the physical renderer, do the following tasks:

- 1 [Setting Up the Physical Renderer on page 84](#)
- 2 [Customizing the Physical Renderer to Render List Applets on page 86](#)
- 3 [Customizing the Physical Renderer to Bind Events on page 88](#)
- 4 [Customizing the Physical Renderer to Bind Data on page 89](#)
- 5 [Customizing the Physical Renderer to Refresh the Recycle Bin on page 89](#)
- 6 [Customizing the Event Handlers on page 92](#)
- 7 [Modifying the CSS Files to Support the Physical Renderer on page 93](#)

In this topic, you customize the JQGridRenderer physical renderer that Siebel Open UI uses with a presentation model for a typical Siebel list applet so that it renders this applet as a grid. You add the rendering capabilities for the carousel that Siebel Open UI uses to render the recycle bin. You also modify the grid style to accommodate the carousel control. You use methods in the physical renderer to do this work. For a description of these methods, including the sequence you use to configure them, see ["Life Cycle of a Physical Renderer" on page 56](#).

Setting Up the Physical Renderer

This task is a step in ["Process of Customizing the Physical Renderer" on page 84](#).

Figure 24 illustrates the code you use to set up the physical renderer. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```

if( typeof( SiebelAppFacade.RecycleBinRenderer ) === "undefined" ){
  SiebelJS.Namespace( "SiebelAppFacade.RecycleBinRenderer" );
  define("siebel/custom/recyclebinrenderer",
    [ "3rdParty/jcarousel/lib/jquery.jcarousel.min", "siebel/jqgridrenderer" ], function () {
    SiebelAppFacade.RecycleBinRenderer = { function(){
      var siebConsts = SiebelJS.Dependency( "SiebelApp.Constants" );
      function RecycleBinRenderer( pm ){
        SiebelAppFacade.RecycleBinRenderer.superclass.constructor.call( this, pm );
        this.listOfCols = [ "Name", "Location" ];
      }
      SiebelJS.Extend( RecycleBinRenderer, SiebelAppFacade.JQGridRenderer );
    }
  }
  return RecycleBinRenderer;
}

```

Figure 24. Setting Up the Physical Renderer

To set up the physical renderer

- 1 Download a copy of the recyclebinrenderer.js file to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

It is recommended that you get a copy of this file to assist in your understanding of how to implement the example that this topic describes. This file includes all the code that this example uses. It also includes more comments that describe code functionality. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

For more information about the folders you can use to store your customizations, see ["Organizing Files That You Customize" on page 122](#).

- 2 Use a JavaScript editor to open the recyclebinmodel.js file that you downloaded in [Step 1](#).
- 3 Verify that the RecycleBinRenderer class does not exist and that you do not configure Siebel Open UI to override this class:

```
if( typeof( SiebelAppFacade.RecycleBinRenderer ) === "undefined" ){
```

- 4 To prevent potential conflicts, create a namespace that Siebel Open UI can use:

```
SiebelJS.Namespace("SiebelAppFacade.RecycleBinRenderer");
```

- 5 Use the Define method to identify the physical renderer file:

```
define("siebel/custom/recyclebinrenderer", [ "3rdParty/jcarousel/lib/
jquery.jcarousel.min", "siebel/jqgridrenderer" ], function () {
```

You must use the Define method to make Siebel Open UI can identify the constructor. You must include the relative path and the name of the presentation model file without the file name extension. For more information, see ["Define Method" on page 506](#).

- 6 Define the class:

```
SiebelAppFacade.RecycleBinRenderer = (function(){
```

- 7 Declare the variables that Siebel Open UI uses throughout the physical renderer code:

```
var siebConsts = SiebelJS.Dependency("Siebel App. Constants");
```

- 8 Create the class constructor:

```
function RecycleBinRenderer(pm){
    SiebelAppFacade.RecycleBinRenderer.superclass.constructor.call(this, pm);
    this.listOfCols = ["Name", "Location"];
}
```

- 9 Define the inheritance chain:

```
SiebelJS.Extend(RecycleBinRenderer, SiebelAppFacade.JQGridRenderer);
```

For more information about inheritance chains, see [“About Dependency Injection” on page 69](#).

- 10 Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Render List Applets

This task is a step in [“Process of Customizing the Physical Renderer” on page 84](#).

The ShowUI method of the JQGridRenderer physical renderer renders a list applet in the JQGrid control. The ShowUI method in the physical renderer that the recycle bin uses places the third-party JCarousel control next to the grid. For more information, see [“ShowUI Method” on page 464](#).

Figure 25 illustrates the code you use to customize the physical renderer to render list applets. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
RecycleBinRenderer.prototype.ShowUI = function(){
    SiebelAppFacade.RecycleBinRenderer.superclass.ShowUI.call( this );
    var pm = this.GetPM();
    var placeholder = "s " + pm.Get( "GetFullId" ) + " div";
    $( "# " + placeholder )
        .addClass( "siebui-list-recyclebin" )
        .after( "<ul id='" + placeholder + " recycle'" class='siebui-list-carousel jcarousel-skin-tango'></ul>" )
        .nextAll( "# " + placeholder + " recycle" )
        .jcarousel({
            scroll          : 10,
            size           : 0,
            vertical       : true,
            itemFallbackDimension : 150
        })
        .data("jcarousel")
        .setup();
},
```

Figure 25. Customizing the Physical Renderer to Render List Applets

To customize the physical renderer to render list applets

- 1 In the recyclebinrenderer.js file, call the ShowUI method of the physical renderer:

```
SiebelAppFacade.RecycleBinRenderer.superclass.ShowUI.call(this);
```

If you customize a physical renderer, then it is recommended that you call each life cycle method of the predefined renderer before you run any custom logic.

- 2 Get the presentation model instance:

```
var pm = this.GetPM();
```

For more information, see ["GetPM Method for Physical Renderers" on page 463](#).

- 3 Calculate the placeholder ID of the HTML node that Siebel Open UI uses as the container for the predefined applet:

```
var placeholder = "s_" + pm.Get("GetFullId") + "_div";
```

You use this ID to modify the HTML DOM. For example, to position the carousel in the recycle bin. The `GetFullId` property gets the unique ID of each applet that is in scope in a view. It uses the following format:

```
s_FullID_div
```

where:

- *FullId* in this example is `S_A1`. The entire ID in this example is `s_S_A1_div`. *FullId* is not a complete ID. It is a part of the ID string template named `s_FullId_div`.

For more information, see ["Properties of the Presentation Model That Siebel Open UI Uses for Applets" on page 436](#).

- 4 Resize the container:

```
$("#" + placeholder)
```

This code attaches the `siebui-list-recyclebin` CSS class to reduce the width of the list applet and to provide sufficient room for the carousel in the recycle bin. This configuration is an example of using a cascading style sheet to dynamically modify how Siebel Open UI displays the HTML.

- 5 Add a CSS class:

```
.addClass("siebui-list-recyclebin")
```

- 6 Add the container node for the carousel after the div container for the main applet, and add CSS classes to the carousel container:

```
.after("<ul id=\"" + placeholder + "_recycle\" class='siebui-list-carousel  
jcarousel-skin-tango' ></ul>")
```

- 7 Get a DOM reference to this new node:

```
.nextAll("#" + placeholder + "_recycle")
```

- 8 Set the size:

```
.jcarousel({  
  scroll: 10,  
  size: 0,  
  vertical: true,  
  itemFallbackDimension: 150
```

- 9 Apply the carousel plug-in to the reference that [Step 7](#) gets:

```
.data('j carousel')
```

- 10 Set up the plug-in:

```
.setup();
```

- 11 Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Bind Events

This task is a step in ["Process of Customizing the Physical Renderer" on page 84](#).

In this topic, you add the following functionality to the carousel:

- If the user hovers the mouse over a record in the carousel, then display a restore button as a plus sign (+).
- If the user removes the hover, then hide the restore button.
- If the user clicks the plus sign (+), then call the presentation model to restore the record.

To add this functionality, you customize Siebel Open UI to attach an event handler to each of the following items:

- To the carousel item for each hover activity.
- To the HTML node that Siebel Open UI uses for the restore button.

To customize the physical renderer to bind events

- 1 In the recyclebinrenderer.js file, call the BindEvents method of the physical renderer:

```
SiebelAppFacade.RecycleBinRenderer.superclass.BindEvents.call(this);
```

For more information, see ["BindEvents Method" on page 461](#).

- 2 Locate the HTML nodes that you must modify.
- 3 Attach event handlers for each event to the nodes that you located in [Step 2](#):

```
$("#s_" + this.GetPM().Get("GetFullId") + "_div")
    .parent()
    .delegate(
        "div.siebui-carousel-item", "mouseenter", {ctx: this}, ShowRestoreButton)
    .delegate(
        "div.siebui-carousel-item", "mouseleave", {ctx: this}, HideRestoreButton)
    .delegate(
        "a.siebui-item-add", "click", {ctx: this}, AddFromRecycleBin);
```

You implement these event handlers in ["Customizing the Event Handlers" on page 92](#). For more information, see ["GetPM Method for Physical Renderers" on page 463](#).

- 4 Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Bind Data

This task is a step in [“Process of Customizing the Physical Renderer” on page 84](#).

The carousel in this example does not render data. Siebel Open UI only renders data in this example if it adds a record to the recycle bin or deletes a record from the recycle bin.

To customize the physical renderer to bind data

- 1 In the recyclebinrenderer.js file, add the following code to SiebelAppFacade.RecycleBinRenderer = (function(){


```
RecycleBinRenderer.prototype.BindData = function(){
    SiebelAppFacade.RecycleBinRenderer.superclass.BindData.apply(this,
    arguments);
  };

```

For more information, see [“BindData Method” on page 461](#).

- 2 Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Refresh the Recycle Bin

This task is a step in [“Process of Customizing the Physical Renderer” on page 84](#).

At this point in this example, you have configured the ShowUI, BindData, and BindEvents methods of the physical renderer, and this renderer displays the carousel with no records. To display deleted records in the carousel, you customize Siebel Open UI to bind the data from these deleted records to the carousel control. To do this, you use dependency injection through the AttachPMBinding method. For more information, see [“About Dependency Injection” on page 69](#) and [“AttachPMBinding Method” on page 428](#).

Siebel Open UI includes the AttachPMBinding method in the presentation model, but it is recommended that you configure Siebel Open UI to call it from the physical renderer so that the presentation model remains independent of methods that you declare in the physical renderer. AttachPMBinding adds a dependency from a physical renderer method to a presentation model method or property. If Siebel Open UI modifies a property value or runs a method in the presentation model, then it uses this dependency to call a method that resides in the physical renderer.

Figure 26 illustrates the code you use to customize the physical renderer to refresh the recycle bin. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
function RecycleBinRenderer( pm ){
    SiebelAppFacade.RecycleBinRenderer.superclass.constructor.call( this, pm );
    this.listOfCols = [ "Name", "Location" ];
}

SiebelJS.Extend( RecycleBinRenderer, SiebelAppFacade.JQGridRenderer );
RecycleBinRenderer.prototype.Init = function () {
    SiebelAppFacade.RecycleBinRenderer.superclass.Init.call( this );
    this.AttachPMBinding( "RefreshList", RefreshCarousel );
};
.
.
.
function RefreshCarousel(){
    var pm          = this.GetPM(),
        recordSet   = pm.Get( "DeletionCompleteSet" ),
        el          = $( "#s_" + pm.Get( "GetFullId" ) + "_div" + "_recycle" ),
        carousel    = el.data( 'jcarousel' ),
        count       = 0;

    carousel.reset();
    for( var i = 0, len = recordSet.length; i < len; i++ ){
        if( recordSet[i] ){
            carousel
                .add( count,
                    "<li>" + GetCurrentCarouselItems.call( this, recordSet[i],
                    this.listOfCols, i ) + "</li>" );
            count++;
        }
    }

    carousel.size( count );
    el.find( "a.siebui-citem-add" ).hide();
    el = carousel = null;
}
}
```

Figure 26. Customizing the Physical Renderer to Refresh the Recycle Bin

To customize the physical renderer to refresh the recycle bin

- 1 In the recyclebinrenderer.js file, bind the RefreshCarousel method that the physical renderer contains to the RefreshList method that the presentation model contains:

```
Siebel JS. Extend(RecycleBinRenderer, Siebel AppFacade. JQGridRenderer);
RecycleBinRenderer.prototype.Init = function () {
    Siebel AppFacade. RecycleBinRenderer.superclass.Init.call(this);
    this.AttachPMBinding("RefreshList", RefreshCarousel);
};
```

In this example, you implemented the RefreshList method in the presentation model in [Step 12 on page 77](#). This presentation model calls the RefreshList method when the user adds a record or removes a record from the recycle bin. AttachPMBinding configures Siebel Open UI to call RefreshCarousel when the presentation model runs the RefreshList method. You must configure your custom physical renderer to call the AttachPMBinding method so that it overrides the Init function. You must make sure you configure Siebel Open UI to call the Init function of the superclass before it creates or attaches a modification in your custom physical renderer.

You must specify all AttachPMBinding calls in the Init function in the physical renderer.

- 2 Configure the RefreshCarousel to read the value of the DeletionCompleteSet property in the physical renderer:

```
var pm = this.GetPM(),
    recordSet = pm.Get("DeletionCompleteSet");
```

- 3 Calculate the container in the HTML DOM that hosts the carousel:

```
el = $("#s_" + pm.Get("GetFullId") + "_div" + "_recycle");
```

- 4 Find the data in the carousel:

```
carousel = el.data('jcarousel');
```

- 5 Reset the data:

```
carousel.reset();
```

- 6 Declare the following variable:

```
this.listOfCols = [ "Name", "Location" ];
```

- 7 Refresh the carousel:

```
var count = 0;
for(var i = 0, len = recordSet.length; i < len; i++){
    if(recordSet[i]){
        carousel
        .add(count,
            "<li>" + GetCurrentCarouselItems.call(this, recordSet[i], this.listOfCols,
            i) + "</li>");
        count++;
    }
}
```

This code does the following work:

- Loops through the set of records that the DeletionCompleteSet property contains.
- Adds the records and the separate items.
- Sends the index of the record that resides in the DeletionCompleteSet property to the GetCurrentCarouselItems method.

- Uses the `GetCurrentCarouselItems` method to create the markup for each carousel item.
- Uses `GetCurrentCarouselItems` to add the index to the markup for the individual item. This configuration makes sure the item is available if the user chooses to restore the record.
- Iterates through a number of list columns according to the `Location` variable that you declare in [Step 6](#).

- 8 Inform the carousel how many items [Step 7](#) added:

```
carousel . si ze(count);
```

This step makes sure scrolling works correctly.

- 9 Hide the restore button on each carousel record:

```
el . fi nd("a. si ebui -ci tem-add"). hi de();
```

- 10 Remove the DOM references:

```
el = carousel = nul l;
```

It is recommended that you remove any DOM references that you configure.

- 11 Save the `recyclebinrenderer.js` file.

Customizing the Event Handlers

This task is a step in [“Process of Customizing the Physical Renderer” on page 84](#).

In this topic, you customize the event handlers that Siebel Open UI uses to handle the state of the restore button, and to handle the event that occurs if the user clicks a record in the recycle bin. You attached these event handlers in [Step 3 on page 88](#).

To customize the event handlers

- 1 In the `recyclebinrenderer.js` file, add the `ShowRestoreButton` method to display the restore button:

```
function ShowRestoreButton(evt){
  if(evt && evt.currentTarget){
    $(evt.currentTarget). chi l dren("a. si ebui -ci tem-add"). show();
  }
}
```

- 2 Add the `HideRestoreButton` method to hide the restore button:

```
function Hi deRestoreButton(evt){
  if(evt && evt.currentTarget){
    $(evt.currentTarget). chi l dren("a. si ebui -ci tem-add"). hi de();
  }
}
```

- 3 Add the `AddFromRecycleBin` method to call the presentation model:

```
function AddFromRecycleBin(evt){
    var pm = evt.data.ctx.GetPM();
    if(evt && evt.currentTarget){
        pm.OnControlEvents("RESTORE",
$(evt.currentTarget).parent().data("index"));
    }
}
```

The AddFromRecycleBin method uses the OnControlEvents method to call the presentation model. It sends the index of the record that the user requests to restore. For more information, see [“OnControlEvents Method” on page 432](#).

- 4 Save the recyclebinrenderer.js file.

Modifying the CSS Files to Support the Physical Renderer

This task is a step in [“Process of Customizing the Physical Renderer” on page 84](#).

In this topic, you modify the CSS files to support the CSS classes that the physical renderer uses.

To modify the CSS files to support the physical renderer

- 1 Open the CSS file.
- 2 Add the following code:

```
.siebui-list-recyclebin{
width: 85%;
float: left;
}

div.jcarousel-skin-tango{
width: 15% !important;
float: left;
text-align: center;
}

div.siebui-carousel-col {
display: block;
}

div.siebui-carousel-item{
height: 75px;
padding: 5px;
border: 1px solid #acacac;
text-align: center;
padding-top: 20px;
word-wrap: break-word;
```

```
font-family: arial ;
font-size: 11px;
}

a.siebui -ci tem-add{
display: block;
/*position: absolute; */
top: 2px;
right: 2px;
float: right;
width: 16px;
height: 16px;
background: url (../images/plus.png) no-repeat center center;
}
```

- 3 Save the CSS file.

Configuring the Manifest for the Recycle Bin Example

This task is a step in ["Roadmap for Customizing Siebel Open UI" on page 61](#).

This topic describes how to configure the manifest for the recycle bin example. For more information, see ["Configuring Manifests" on page 128](#).

To configure the manifest for the recycle bin example

- 1 Make sure your presentation model and physical renderer uses the define method.
You do this in [Step 4 on page 63](#) for the presentation model and in [Step 5 on page 85](#) for the physical renderer.
- 2 Configure the manifest:
 - a Log in to the Siebel client with administrative privileges.
 - b Navigate to the Administration - Application screen, and then the Manifest Files view.
 - c In the Files list, add the following new files.

| Field | Value |
|-------|-------------------------------------|
| Name | siebel/custom/recyclebinrenderer.js |
| Name | siebel/custom/recyclebinmodel.js |

- d Navigate to the Administration - Application screen, and then the Manifest Administration view.

- e In the UI Objects list, specify the following applet.

| Field | Value |
|------------|-------------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | SIS Account List Applet |

- f In the Object Expression list, add the following expression. The physical renderer uses this expression to render the applet in a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 1 |

- g In the Files list, add the following file:
siebel/custom/recyclebinrenderer.js

- h In the UI Objects list, specify the following applet.

| Field | Value |
|------------|-------------------------|
| Type | Applet |
| Usage Type | Presentation Model |
| Name | SIS Account List Applet |

- i In the Object Expression list, add a record with no value in the Expression field.

- j In the Files list, add the following file:
siebel/custom/recyclebinmodel.js

Testing Your Modifications

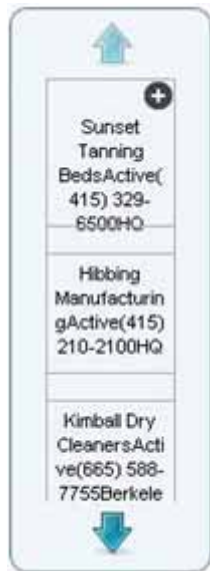
This task is a step in [“Roadmap for Customizing Siebel Open UI” on page 61](#).

In this topic, you test your modifications.

To test your modifications

- 1 Log in to the Siebel Open UI client, and then navigate to the Accounts screen.
- 2 Use the Select column to choose five account records, and then click Delete.
Siebel Open UI deletes the records and adds them to the carousel recycle bin.

- 3 To restore a record, click the following plus (+) icon in the carousel recycle bin:



- 4 Verify that Siebel Open UI recreates the record on the Siebel Server and adds it back to the Account list.

5

Customizing Siebel Open UI

This chapter describes how to customize Siebel Open UI. It includes the following topics:

- [Guidelines for Customizing Siebel Open UI on page 97](#)
- [Doing General Customization Tasks on page 100](#)
- [Managing Files on page 122](#)
- [Configuring Manifests on page 128](#)

Guidelines for Customizing Siebel Open UI

This topic describes guidelines for configuring Siebel Open UI. It includes the following information:

- [Guidelines for Customizing Presentation Models on page 97](#)
- [Guidelines for Customizing Physical Renderers on page 99](#)
- [Guidelines for Customizing Presentation Models and Physical Renderers on page 100](#)

Some Siebel Open UI customizations use the same configuration that a Siebel Business Application uses. For example, you can use the information that *Configuring Siebel Business Applications* describes to configure the following items in Siebel Open UI:

- List applets
- Form applets
- Views that contain more than one applet
- Applet controls and list columns

Guidelines for Customizing Presentation Models

It is recommended that you apply the following guidelines if you configure a presentation model:

- Make sure you customize Siebel Open UI so that the user-interface state is separate from the rendering of this state. The guidelines in this topic describe how to do this.

- Add a new presentation model only after you consider all other customization options, such as modifying code in a Siebel Web Template file or using Siebel Tools to modify an object. To examine some examples that do not modify the presentation model, see [Chapter 5, “Customizing Siebel Open UI.”](#)

A presentation model implements the entire abstraction of the user interface content, so the predefined implementation for a presentation model implements the predefined abstraction. There are only a few types of basic user interface abstractions, such as single record, list, tree, and so on. It is recommended that you use a predefined presentation model for each of these basic abstractions that Oracle provides you. The only situation where it is recommended that you use the predefined presentation model is for a new predefined abstraction. For example, to implement a graph abstraction for a social network model.

- Make sure Siebel Open UI models all the state variables that it requires to achieve a rich client behavior, and that it models these state variables as presentation model properties. These properties can reside in the presentation model on the client, or the Siebel Server can provide them from an applet. You can add methods that modify these properties and that manage the state changes after you configure Siebel Open UI to add them. Siebel Open UI typically calls these methods due to a user action or if the server sends a notification. If a method modifies the logical state of the user interface, then Siebel Open UI uses the `AttachPMBinding` method to add a binding trigger to the physical renderer. This trigger binds the modified state to the physical user interface. For more information, see [“AttachPMBinding Method” on page 428.](#)

Siebel Open UI strictly defines each life cycle method. To help make sure your implementation is clean and readable, it is recommended that you use the following guidelines:

- Make sure Siebel Open UI uses all presentation model state variables as properties. You must use the `AddProperty` method to create these properties. You must not use ordinary JavaScript variables to create these properties.
- Use methods to implement all state changes of the presentation model. Use the `AddMethod` method to create these methods.
- Make sure Siebel Open UI uses the `AttachEventHandler` method to bind each method that the presentation model contains to an event that the physical renderer contains. Each event is the result of some physical user action. This configuration makes sure Siebel Open UI binds each user action to the required logic and modifies the user interface state. For more information, see [“AttachEventHandler Method” on page 427.](#)
- A presentation model method can start a call to the Siebel Server, and then the server sends a reply to this method. Siebel Open UI handles these calls asynchronously, except for high interactivity Siebel browser scripts. For more information, see [“About Synchronous and Asynchronous Requests” on page 74.](#)
- Siebel Open UI includes all modifications that occur in the business component layer in the reply in a Notification property set. You must use the `AttachNotificationHandler` method to add this notification. For more information, see [“Notifications That Siebel Open UI Supports” on page 541.](#)
 - Siebel Open UI packages a reply from the server for any predefined type of request. It includes this package in a reply property set that is predefined. You must use the `AttachPSHandler` method to add the handler for any property set type that the server sends.

- You must use the `AttachPostProcessingHandle` method to add any post-processing handler that does follow up logic on a server request, such as a `NewRecord` request. You can add this logic after Siebel Open UI finishes processing the reply for this request. Setting the focus for a control is an example of this kind of configuration.
- Siebel Open UI does the initial setup of the presentation model when it initializes the Siebel view or application, depending on whether the user interface object resides in or outside of the view. The server sends a property set that includes all the initialization attributes. The proxy uses most of these attributes, but you must use the `AddProperty` method to get the values that the presentation model requires to set and store the state.
- You must use the following methods in the physical renderer the first time Siebel Open UI renders the user interface:
 - **BindEvents.** Binds the presentation model methods to the appropriate physical events on the physical control. For more information, see [“BindEvents Method” on page 461](#).
 - **BindData.** Accesses all the properties in the presentation model and sends them to the physical control through the appropriate methods that this physical control exposes. For more information, see [“BindData Method” on page 461](#).
- You must configure Siebel Open UI to bind any state changes to the presentation model that occur after the physical renderer finishes the initial rendering. To do this, you configure Siebel Open UI to call the `AttachPMBinding` method on the physical renderer. This configuration specifies the method that the physical renderer must call or the properties that it must access so that it can send data back to the physical control. This configuration allows Siebel Open UI to render the user interface after it modifies the presentation model state.

Guidelines for Customizing Physical Renderers

It is recommended that you apply the following guidelines if you configure a physical renderer:

- Use a physical renderer only to implement methods that render the presentation model state:
 - Do not include any other logic in a physical renderer.
 - Do not include business logic that modifies the user interface state.
 - Only use a physical renderer to send user action events to the presentation model, and use the presentation model to do all the work that is necessary to modify a state.
 - Allow the physical renderer to rebind the new presentation model state to the rendered user interface only after the presentation model finishes modifying the state of the logical user interface.
- Do not use a physical renderer to add any presentation attributes to the Document Object Model (DOM). Example attributes include position, color, or any other styling. To modify a presentation attribute, you must attach or detach a style that you define in a CSS file.
- Configure Siebel Open UI to do all rendering only in a physical renderer. It is strongly recommended that you do not configure Siebel Open UI to do direct DOM manipulation. If you cannot avoid direct DOM manipulation, then you must do this manipulation in a physical renderer. Configure Siebel Open UI to send data, metadata, or state information to controls only from a physical renderer. For more information, see [“About Objects and Metadata” on page 33](#).

- In most situations, if you add a presentation model, then you must also add a corresponding physical renderer. You typically use a presentation model to add custom logic in the client. This logic typically determines a physical behavior that requires a physical renderer to do the rendering. For example, in most situations, you cannot configure a predefined applet that also renders custom logic. Siebel Open UI structures custom JavaScript logic in the presentation model and physical renderer as a customization of predefined Siebel Open UI. This structure allows Siebel Open UI to use Oracle JavaScript and to use other logic that a predefined Siebel Open UI provides, such as events, Siebel arrays, and so on. It is not recommended that you configure JavaScript that is independent of Siebel Open UI, and that also modifies Siebel CRM data or physical behavior.

Guidelines for Customizing Presentation Models and Physical Renderers

It is recommended that you apply the following guidelines if you configure the presentation model and physical renderer for a client object:

- Determine the following items for any element that you intend to customize:
 - The presentation model you must use
 - The physical user interface control you must use and the physical renderer that you must use with the presentation model
- Configure the manifest so that Siebel Open UI can identify the JavaScript files it must download to the client so that it can render the user interface element. For more information, see [“Configuring Manifests” on page 128](#).
- Modify the physical renderer and presentation model for user interface objects that do not reside in a view, such as navigation tabs. Only one of these elements resides on a single Siebel page and they do not vary during a Siebel session. So, you can configure the physical renderer and the presentation model for each of these elements in the manifest.
- You must place all custom presentation models and physical renderers in a custom folder. For more information about this folder, see [“Organizing Files That You Customize” on page 122](#).

Doing General Customization Tasks

This topic describes some of the general customization tasks that you can do in Siebel Open UI. It includes the following information:

- [Enabling Object Managers for Siebel Open UI on page 101](#)
- [Preparing Siebel Tools to Customize Siebel Open UI on page 104](#)
- [Modifying the Application Configuration File on page 105](#)
- [Adding Presentation Model Properties That Siebel Servers Send to Clients on page 106](#)
- [Configuring Siebel Open UI to Bind Methods on page 110](#)
- [Calling Methods for Applets and Business Services on page 111](#)

- [Using the Base Physical Renderer Class With Nonapplet Objects on page 113](#)
- [Customizing Events on page 118](#)
- [Creating Components on page 119](#)
- [Allowing Users to Interact with Clients During Business Service Calls on page 120](#)

Enabling Object Managers for Siebel Open UI

You must enable the object manager to use Siebel Open UI before you can do any customization work.

To enable object managers for Siebel Open UI

- 1 Enable the object manager for the Siebel Server:
 - a Log in to the Siebel CRM client with administrator privileges.
 - b Navigate to the Administration - Server Configuration screen, and then the Servers view.
 - c In the Components list, query the Component field for the object manager where you must enable Siebel Open UI.

For example, query for the following value:

Call Center Object Manager (ENU)

- d In the bottom applet, click Parameters.
- e In the Component Parameters list, query the Parameter field for EnableOpenUI.
- f Set the Value on Restart field to TRUE.
- g Log out of the client and close the browser.

As an alternative to using the administrative screens, you can modify the application configuration file on the Siebel Server in the same way that you modify this file on the client in [Step 2](#).

- 2 Enable Siebel Open UI on the Mobile Web Client:
 - a On the client computer, use a text editor to open the application configuration file.

For example, to open the configuration file for Siebel Call Center, navigate to the following folder, and then open the uagent.cfg file:

`client_install_location\bin`

- b In the InfraUIFramework section, set the EnableOpenUI parameter to the following value:


```
[InfraUIFramework]
EnableOpenUI=TRUE
```
- c Save, and then close the application configuration file.

- d** Log in to Siebel Call Center and verify that it opens the Siebel Open UI client.

To revert to the high-interactivity client, you can set the EnableOpenUI parameter to FALSE. If you do this reversion, then you must use Internet Explorer, version 8 or earlier when you open the client.

3 Stop the Siebel Server:

- a** On the computer where the Siebel Server resides, click the Start menu, Control Panel, Administrative Tools, and then the Services menu item.
- b** In the Services dialog box, right-click the Siebel Server service, and then click Stop.
- c** Wait for the Siebel Server to stop.

4 Optional. Add object managers for Siebel Mobile:

- a** Stop the Gateway Server, and then make a backup copy of the siebns.dat file.
- b** Restart the Gateway Server.
- c** Set the SIEBEL_HOME environment variable to the following folder:
`SES_HOME/si ebsrvr`
- d** Open Windows Explorer, and then navigate to the following folder:
`SES_HOME/si ebsrvr/bi n/ l anguage_code`
- e** Run the new_compdef_sia.ksh script or the new_compdef_sia.bat file. Use the following parameters:
`./create_new_compdef_sia.ksh GATEWAY: port_number ENTERPRI SE user_name user_password l anguage_code`
- f** Examine the svrcfg log files and make sure Siebel CRM did not log any errors when it enabled the new server components.
- g** Open the Server Manager, and then run the following commands:
`enabl e compgrp Handhel dSync for server server_name`
`enabl e compgrp Handhel dSyncSI S for server server_name`

5 Start the Siebel Server:

- a** Click the Start menu, Control Panel, Administrative Tools, and then the Services menu item.
- b** In the Services dialog box, right-click the Siebel Server service, and then click Start.
- c** Wait for the server to start.
- d** Optional. Make sure the server components you enabled [Step g on page 102](#) are on line.

6 Optional. Add virtual directories for Siebel Mobile object managers:

- a** Use Windows Explorer to navigate to the following folder:
`eappweb_I NSTALL\bi n\`
- b** Create a back up copy of each of the following files:
 - eapp.cfg.

- eapps_sia.cfg.
- Web server configuration files. For example, obj.conf, httpd.conf, and so on.
- c** Stop the HTTP server.
- d** Open Windows Explorer, and then navigate to the following folder:
eappweb_HOME/bin
- e** Create a back up copy for each of the following files:
 - eapps.cfg.
 - eapps_sia.cfg.
 - Web server configuration file.
- f** Navigate to the following folder:
EAPPWEB_HOME/config
- g** Run the new_virdirs.sh script or the new_virdirs.bat script. Use values from the following table:

| Operating System | Command |
|------------------|--|
| Windows | Run the following command: new_virdirs.bat <i>language_code</i> |
| Unix | Run the following command: newvirdirs.sh <i>languages web_server</i> where: <ul style="list-style-type: none"> ■ <i>languages</i> is a list of language code where a comma separates each code ■ <i>web_server</i> identifies the folder where the Web server resides |

- h** If the script fails to run correctly, then you must restore all files from the backup copies you made in [Step b](#) and [Step e](#), and then run the script again until it successfully finishes.
- i** Restart the HTTP Server.
- j** Make sure you can use the URL to access the new server components.
- 7** Log in to the Siebel CRM client and make sure it displays the Siebel Open UI client.

How Siebel Open UI Loads the Siebel Application Depending on How You Set the EnableOpenUI Parameter

Siebel Open UI loads a Siebel application differently depending on how you set the EnableOpenUI parameter:

- EnableOpenUI=FALSE:

- If you customized the default Siebel Web Template, then Siebel Open UI loads the application from the following folder:

`si ebsrvr\webtempl \custom`

- If you did not customize the default Siebel Web Template, then Siebel Open UI finds no file in the `si ebsrvr\webtempl \custom` folder, and it loads the Siebel application from the following default folder:

`si ebsrvr\webtempl`

- **EnableOpenUI=TRUE:**

- If you customized the Siebel Open UI Siebel Web Template, then Siebel Open UI loads the Siebel application from the following folder:

`si ebsrvr\webtempl \oui webtempl \custom`

- If you did not customize the Siebel Open UI Siebel Web Template, then Siebel Open UI finds no file in the `si ebsrvr\webtempl \oui webtempl \custom` folder, and it loads the Siebel application from the following folder:

`si ebsrvr\webtempl \oui webtempl`

- If you did not customize the Siebel Open UI Siebel Web Template, and if Siebel Open UI does not find a file in the `si ebsrvr\webtempl \oui webtempl` folder, then it loads the Siebel application from the following default folder:

`si ebsrvr\webtempl`

Preparing Siebel Tools to Customize Siebel Open UI

This topic describes how to prepare Siebel Tools so that you can use it to customize Siebel Open UI. For more information, see *Using Siebel Tools*.

To prepare Siebel Tools to customize Siebel Open UI

- 1 Add the EnableOpenUI parameter to the Siebel Tools configuration file:

- a In Windows Explorer, navigate to the following folder:

`SI EBEL_TOOLS_HOME\bin\language_code`

- b Use a text editor to open the tools.cfg configuration file.
- c Add the following parameter to the InfraUIFramework section:

`EnableOpenUI = TRUE`

- 2 Display object types:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b Choose the View menu, and then the Options menu item.

- c Click the Object Explorer tab.
- d Scroll down through the Object Explorer Hierarchy window to locate the object type you must display.

To display an object type and all child object types of this object type, make sure it includes a check mark with a white background. Display the Applet object type and all child objects of the Applet object type.

- e Click OK.

Modifying the Application Configuration File

You can use the configuration file to specify parameters that determine how a specific Siebel application runs. For more information about the application configuration file, see *Configuring Siebel Business Applications*.

To modify the application configuration file

- 1 On the client computer, open Windows Explorer, and then navigate to the following folder:

`ORACLE_HOME\client\bin\language_code`

- 2 Use a text editor to open the application configuration file that you must modify.

Each Siebel application uses a different configuration file. For example, Siebel Call Center uses the `uagent.cfg` file. The application configuration file uses the `.cfg` file extension.

- 3 Locate the section that you must modify.

Each application configuration file uses square brackets to indicate a section. For example:

`[InfraUI Framework]`

- 4 Modify an existing parameter or add a separate line for each parameter you must specify.

Use the following format:

`parameter_name = "<param1 param2>"`

where:

- `param1` and `param2` are the names of the parameters.

For example:

`TreeNodeCollapseCaption = ""`

Adding Presentation Model Properties That Siebel Servers Send to Clients

This topic describes how to add presentation model properties that the Siebel Server sends to the client. It includes the following information:

- [“Adding Presentation Model Properties That Siebel Servers Send for Applets” on page 106](#)
- [“Adding Presentation Model Properties That Siebel Servers Send for Views” on page 107](#)
- [“Customizing Control User Properties for Presentation Models” on page 108](#)

It is strongly recommended that you configure custom presentation model properties only if the predefined presentation model properties do meet your requirements.

Adding Presentation Model Properties That Siebel Servers Send for Applets

This topic describes a general approach to customizing applet user properties for presentation models. The Siebel Server sends these properties to the client.

To add presentation model properties that Siebel Servers send for applets

- 1 Add user properties to the applet:
 - a Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b In the Object Explorer, click Applet.
 - c In the Applets list, query the Name property for the applet that you must modify.
For example, query the Name property for Contact List Applet.
 - d In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
 - e In the Applet User Props list, add the following applet user property.

| Name | Value |
|-----------------------------------|--|
| ClientPMUserPropn | <i>user_property_name</i> |
| For example, ClientPMUserProp1 | <p>You can specify one or more user properties. Siebel Open UI sends these user properties to the presentation model that it uses in the client to display the applet. To specify more than one user property, use a comma and a space to separate each user property name. For example:</p> <p style="text-align: center;">User Property1, User Property2</p> <p>Each user property that you specify must exist in the Siebel repository, and each of these user properties must contain a value in the Value property.</p> |

- f** (Optional) Specify more ClientPMUserPropn user properties, as necessary.

You can specify more than one ClientPMUserPropn user property, as necessary. Repeat [Step e](#) for each ClientPMUserPropn user property that you require.

- g** Compile your modifications.

2 Modify the presentation model:

- a** Use a JavaScript editor to open your custom presentation model file that Siebel Open UI uses to display the applet that you modified in [Step 1](#).

- b** If your custom presentation model does not override the Setup method, then configure Siebel Open UI to do this override.

For more information about how to configure an override, see ["Process of Customizing the Presentation Model" on page 62](#).

- c** Locate the following section of code:

```
presentation_model.Setup(propSet)
```

For example, if the class name is CustomPM, then locate the following code:

```
CustomPM.prototype.Setup = function (propSet)
```

- d** Add the following code to the section that you located in [Step c](#):

```
var consts = Siebel JS.Dependency("Siebel App. Constants");
var apm = propSet.GetChildByType(consts.get("SWE_APPLET_PM_PS"));
```

where:

- SWE_APPLET_PM_PS is a predefined constant that Siebel Open UI uses to get the presentation model properties that it uses to display the applet. The Siebel Server sends these properties in a property set.

- e** Add the following code anywhere in the presentation model:

```
var value = apm.GetProperty("user_property_name")
```

For example:

```
var value = apm.GetProperty("User Property1")
```

You must configure Siebel Open UI so that it runs the Setup method that you specify in [Step c](#) before it encounters the code that you add in [Step e](#).

Adding Presentation Model Properties That Siebel Servers Send for Views

This topic describes how to customize view user properties for presentation models. The Siebel Server sends these properties to the client.

To add presentation model properties that Siebel Servers send for views

- 1** Add user properties to the view:

- a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click View.
 - c** In the Views list, query the Name property for the view that you must modify.
For example, query the Name property for Contact List View.
 - d** In the Object Explorer, expand the View tree, and then click View User Prop.
 - e** Do [Step e on page 106](#) through [Step g on page 107](#), except add view user properties to a view instead of adding applet user properties to an applet.
- 2** If your custom view presentation model does not override the Setup method, then configure Siebel Open UI to do this override:
Do [Step 2 on page 107](#) except use `vpm` instead of `apm`:
 - a** Use a JavaScript editor to open the presentation model file that Siebel Open UI uses to display the view that you modified in [Step 1](#).
 - b** Add the following code:

```
var consts = Siebel JS. Dependency("Siebel App. Constants");  
var vpm = propSet.GetChildByType(consts.get("SWE_VIEW_PM_PS"));
```

where:
 - `SWE_VIEW_PM_PS` is a predefined constant that Siebel Open UI uses to get the presentation model properties that it uses to display the view. The Siebel Server sends these properties in a property set.
 - c** Add the following code:

```
var value = vpm.GetProperty("user_property_name")
```

For example:

```
var value = vpm.GetProperty("User Property1")
```

For more information about how to configure an override, see ["Process of Customizing the Presentation Model" on page 62](#).

Customizing Control User Properties for Presentation Models

This topic describes how to customize control user properties for a presentation model.

To customize control user properties for presentation models

- 1** Add user properties to the control:
 - a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click Applet.

- c** In the Applets list, query the Name property for the applet that you must modify.
For example, query the Name property for Contact List Applet.
 - d** In the Object Explorer, expand the Applet tree, and then Control.
 - e** In the Controls list, query the Name property for the control that you must modify.
For example, query the Name property for NewRecord.
 - f** In the Object Explorer, expand the Control tree, and then click Control User Prop.
 - g** In the Control User Props list, Do [Step e on page 106](#) through [Step g on page 107](#), except add control user properties to the control instead of adding applet user properties to an applet.
- 2** Modify the custom presentation model of the applet where the control resides:
- a** Configure Siebel Open UI to get the control object. You can do one of the following:
 - ❑ Use the following code to get the control object from the GetControls presentation model property:


```
var controls = this.Get("GetControls");
for (var control in controls){
    var cpm = control.GetPMPPropSet(consts.get("SWE_CTRL_PM_PS"));
    // Do something with cpm
}
```
 - ❑ Use the following the GetControl method to get an instance of the Account Name control:


```
var myControl = this.GetControl ("Account Name");
var cpm = control.GetPMPPropSet(consts.get("SWE_CTRL_PM_PS"));
```
 - b** Add the following code:


```
var consts = Siebel JS.Dependency("Siebel App. Constants");
var cpm = control.GetPMPPropSet(consts.get("SWE_CTRL_PM_PS"));
```

where:

 - ❑ GetPMPPropSet is a method that gets the property set for this control. For more information, see ["GetPMPPropSet Method" on page 471](#).
 - ❑ SWE_CTRL_PM_PS is a predefined constant that Siebel Open UI uses to get the presentation model that it uses for the control object. The Siebel Server sends these properties in a property set.
 - c** Add the following code:


```
var value = cpm.GetProperty("user_property_name")
```

For example:

```
var value = cpm.GetProperty("User Property1")
```

Configuring Siebel Open UI to Bind Methods

This topic includes some examples that describe how to bind methods. For other examples that bind methods, see the following topics:

- ["Example of the Life Cycle of a User Interface Element" on page 58](#)
- ["Customizing the Physical Renderer to Refresh the Recycle Bin" on page 89](#)
- ["Text Copy of Code That Does a Partial Refresh for the Physical Renderer" on page 162](#)

Binding Methods That Reside in the Physical Renderer

You can use the AttachPMBinding method to bind a method that resides in a physical renderer and that Siebel Open UI must call when the presentation model finishes processing.

To bind methods that reside in the physical renderer

- 1 Add the method reference in the physical renderer.
- 2 Configure Siebel Open UI to send the scope in the binderConfig argument of the AttachPMBinding method as a scope property.

For more information, see ["AttachPMBinding Method" on page 428](#).

Conditionally Binding Methods

The example in this topic conditionally binds a method.

To conditionally bind methods

- Add the following code:

```
this.AttachPMBinding("DoSomething", function(){SiebelJS.Log("After
DoSomething");},{when: function(function_name){return false;}});
```

where:

- *function_name* identifies the name of a function.

{In this example, if Siebel Open UI calls DoSomething, then the presentation model calls the *function_name* that the when condition specifies, and then tests the return value. If *function_name* returns a value of:

- **true.** Siebel Open UI calls the AttachPMBinding method.
- **false.** Siebel Open UI does not call the AttachPMBinding method.

If you do not include the when condition, then Siebel Open UI runs the DoSomething method, and then calls the AttachPMBinding method. For more information, see ["AttachPMBinding Method" on page 428](#).

Calling Methods for Applets and Business Services

This topic includes some examples that describe how to call methods for applets and business services. For other examples that call methods, see the following topics:

- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Attaching an Event Handler to a Presentation Model” on page 78](#)
- [“Using Custom JavaScript Methods” on page 364](#)
- [“Using Custom Siebel Business Services” on page 367](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)

Calling Methods for Applets

The example in this topic describes how to call a method for an applet when the user clicks a button.

To call methods for applets

1 Modify the physical renderer:

- a** Use a JavaScript editor to open the physical renderer for the applet that you must modify.
- b** Locate the onclick code for the button that you must modify.
- c** Add the following code to the code you located in [Step b](#):

```
var inPropSet = CCFMi sCUtil _CreatePropSet();
//Define the inPropSet property set with the information that InvokeMethod
sends as input to the method that it calls.
var ai = {};
ai.async = true;
ai.selfbusy = true;
ai.scope = this;
ai.mask = true;
ai.opdecode = true;
ai.errcb = function(){
    //Code occurs here for the method that Siebel Open UI runs if the AJAX call
    fails
};
ai.cb = function(){
    //Code occurs here for the method that Siebel Open UI runs if the AJAX call
    is successful
};
this.GetPM().ExecuteMethod("InvokeMethod", input arguments, ai);
```

where:

- *input arguments* lists the arguments that InvokeMethod sends as input to the method that it calls.

For example, the following code specifies to use the InvokeMethod method to call the NewRecord method, using the properties that the inPropSet variable specifies for the ai argument:

```
this.s.GetPM().ExecuteMethod("InvokeMethod", "NewRecord", inPropSet, ai);
```

For more information, see [“InvokeMethod Method for Application Models” on page 485](#) and [“NewRecord Method” on page 474](#).

2 Modify the presentation model:

- a Use a JavaScript editor to open the presentation model for the applet that you must modify.
- b Locate the code that calls the Init method.
- c Add the following code to the code that you located in [Step b](#):

```
this.s.AttachPreProxyExecuteBinding("method_name", function(methodName,
inputPS, outputPS){// Include code here that Siebel Open UI runs before the
applet proxy sends a reply. });
```

```
this.s.AttachPostProxyExecuteBinding("method_name", function(methodName,
inputPS, outputPS){// Include code here that Siebel Open UI runs after the
applet proxy sends a reply. });
```

where:

- *method_name* identifies the name of the method that InvokeMethod calls. Note that Siebel Open UI comes predefined to set the value of the methodName argument in the following code to WriteRecord, by default. You must not modify this argument:

```
function(methodName, inputPS, outputPS)
```

For example:

```
this.s.AttachPreProxyExecuteBinding("WriteRecord", function(methodName,
inputPS, outputPS){// Include code here that Siebel Open UI runs before the
applet proxy sends a reply. });
```

```
this.s.AttachPostProxyExecuteBinding("WriteRecord", function(methodName,
inputPS, outputPS){// Include code here that Siebel Open UI runs after the
applet proxy sends a reply. });
```

For more information, see [“WriteRecord Method” on page 407](#), [“AttachPostProxyExecuteBinding Method” on page 429](#), and [“AttachPreProxyExecuteBinding Method” on page 430](#).

Calling Methods for Business Services

The example in this topic describes how to call a method for a business service when the user clicks a button.

To call methods for business services

- 1 Use a JavaScript editor to open the physical renderer for the applet that you must modify.
- 2 Locate the onclick code for the button that you must modify.
- 3 Add the following code to the code you located in [Step 2](#):


```

var service = SiebelApp.S_App.GetService("business_service_name");
if (service) {
    var inPropSet = CCFMiscUtil.CreatePropSet();
    //Code occurs here that sets the inPropSet property set with all information
    that Siebel Open UI must send as input to the method that it calls.
    var ai = {};
    ai.async = true;
    ai.selfbusy = true;
    ai.scope = this;
    ai.mask = true;
    ai.opdecode = true;
    ai.errcb = function(){
        //Code occurs here for the method that Siebel Open UI runs if the AJAX call
        fails
    };
    ai.cb = function(){
        //Code occurs here for the method that Siebel Open UI runs if the AJAX call
        is successful
    };
    service.InvokeMethod("method_name", "input_arguments", ai);
}

```

For more information, see ["InvokeMethod Method for Application Models" on page 485](#).

Using the Base Physical Renderer Class With Nonapplet Objects

This topic describes how to use the Base Physical Renderer class with nonapplet objects that you customize. It includes the following topics:

- [Hierarchy That the Base Physical Renderer Class Uses on page 114](#)
- [Modifying Nonapplet Configurations for Siebel CRM Version 8.1.1.10, 8.2.2.3, or Earlier on page 118](#)
- [Declaring the AttachPMBinding Method When Using the Base Physical Renderer Class on page 116](#)
- [Sending an Arbitrary Scope on page 117](#)
- [Accessing Proxy Objects on page 117](#)

The BasePhysicalRenderer class simplifies calls that Siebel Open UI makes to the AttachPMBinding method for nonapplet objects. You can configure Siebel Open UI to use the BasePhysicalRenderer class to identify the physical renderer, call AttachPMBinding, and specify the configuration for the scope of a nonapplet object. You can then use a custom physical renderer to call AttachPMBinding with the appropriate handler.

Siebel Open UI uses the PhysicalRenderer class to interface with and to render applets. Starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4, it uses the BasePhysicalRenderer class to render nonapplet objects. It uses this class to separate the interface to the physical renderer from the physical renderer. Siebel Open UI uses the BasePhysicalRenderer class only with nonapplet objects, such as the toolbar or predefined query bar.

If your deployment includes nonapplet custom rendering, and if it uses Siebel CRM version 8.1.1.10, 8.2.2.3 or earlier, then it is strongly recommended, but not required, that you modify your configuration so that it uses the `BasePhysicalRenderer` class to render your custom, nonapplet objects. If your deployment uses the `PhysicalRenderer` class to render nonapplet objects, then this class will provide access to applet functionality and properties that it does not require to do the rendering, which could degrade performance or result in rendering problems.

Siebel Open UI defines the `BasePhysicalRenderer` class in the `basephyrenderer.js` file.

Hierarchy That the Base Physical Renderer Class Uses

Figure 27 illustrates the hierarchy that the `BasePhysicalRenderer` class uses for nonmobile applications. The *member variable* is a variable that is associated with the class. All methods can access this member variable.

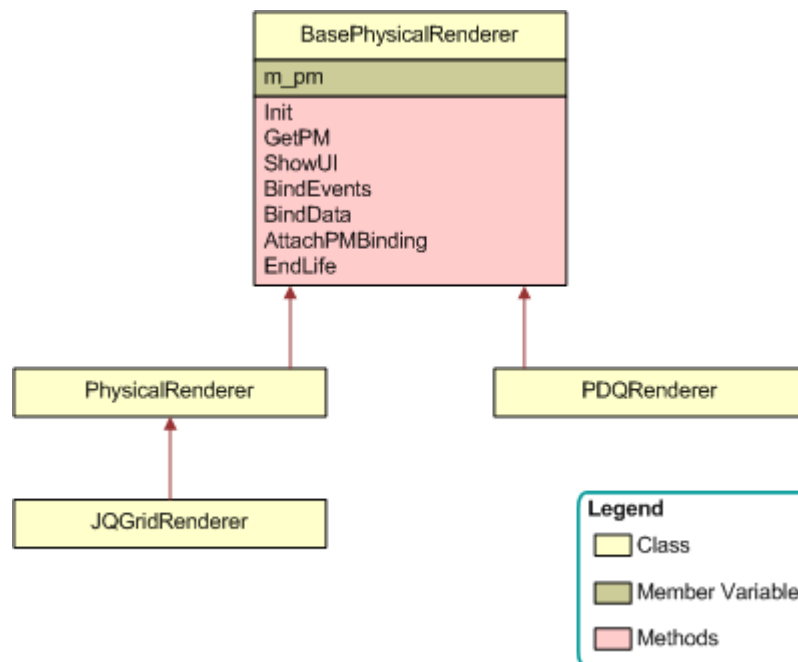


Figure 27. Hierarchy That the Base Physical Renderer Class Uses

Figure 28 illustrates the hierarchy that the BasePhysicalRenderer class uses for mobile applications.

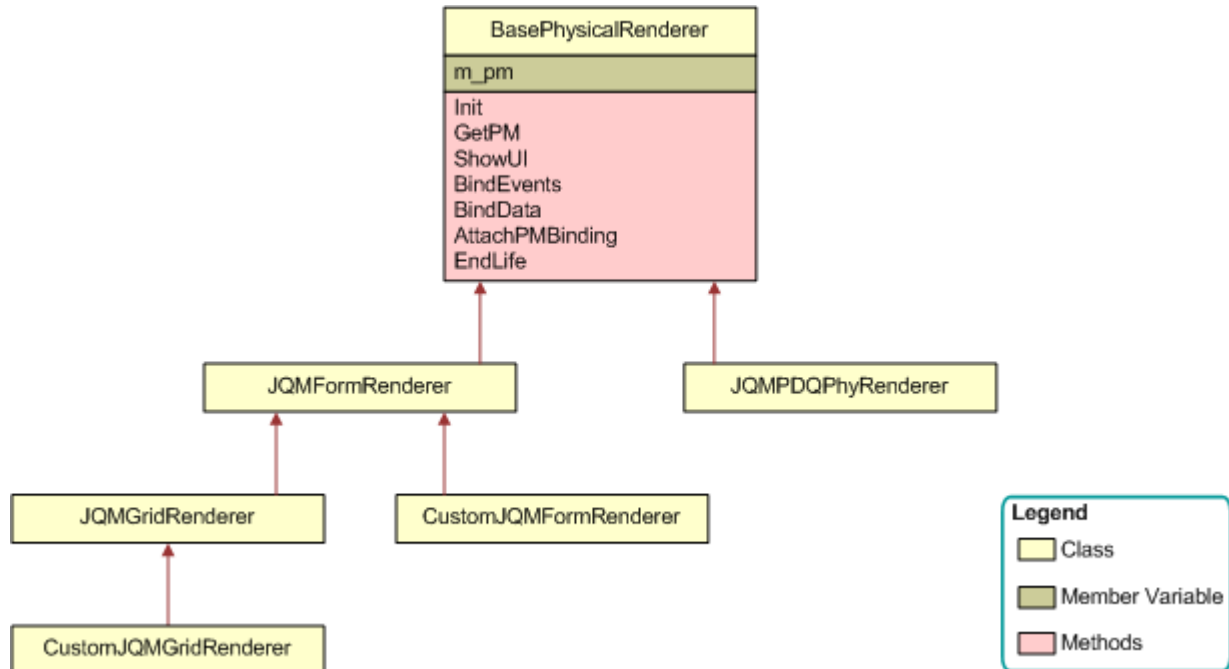


Figure 28. Hierarchy That the Base Physical Renderer Class Uses for Mobile Applications

Using Methods with the Base Physical Renderer Class

Table 6 describes how to use methods with the BasePhysicalRenderer class.

Table 6. How to Use Methods with the Base Physical Renderer Class

| Method | Description |
|------------|---|
| Init | Use this method to initialize the BasePhysicalRenderer class. For more information, see "Init Method" on page 431 . |
| GetPM | Use this method to return the name of the physical renderer. For more information, see "GetPM Method for Physical Renderers" on page 463 . |
| ShowUI | Use this method to display each control in a derived class. You can configure Siebel Open UI to override this method if you must modify the control of the user interface object. For more information, see "ShowUI Method" on page 464 . |
| BindEvents | Use this method to attach an event to the physical control. You can configure Siebel Open UI to override this method if you must modify event binding. For more information, see "BindEvents Method" on page 461 . |

Table 6. How to Use Methods with the Base Physical Renderer Class

| Method | Description |
|-----------------|--|
| BindData | Use this method to bind data to a physical control object that resides in a derived class. You can configure Siebel Open UI to override this method if you must modify the data binding. For more information, see “BindData Method” on page 461 . |
| AttachPMBinding | <p>Use this method to configure Siebel Open UI to do the same work that the AttachPMBinding method does in a presentation model. You can use the following argument to call the AttachPMBinding method:</p> <p>scope</p> <p>You can use the following arguments with the AttachPMBinding method:</p> <ul style="list-style-type: none"> ■ methodName. Identifies the method that the BasePhysicalRenderer class binds. ■ handler. Identifies the handler method for this binding. ■ handlerScope. Identifies the scope where the BasePhysicalRenderer class runs the handler. If you do not specify the handlerScope, then the BasePhysicalRenderer class uses the default scope. <p>For more information, see “AttachPMBinding Method” on page 428.</p> |
| EndLife | Use this method to end the life of the physical renderer. It is recommended that you use the EndLife method to clean up the custom event handler. This clean up includes releasing events, deleting unused variables, and so on. For more information, see “EndLife Method” on page 462 . |

Declaring the AttachPMBinding Method When Using the Base Physical Renderer Class

If you configure Siebel Open UI to use the BasePhysicalRenderer class, then you must declare the AttachPMBinding method.

To declare the AttachPMBinding method when using the Base Physical Renderer class

- 1 Use a JavaScript editor to open your custom physical renderer.
- 2 Locate the Init method.
- 3 Add the following code to the Init method that you located in [Step 2](#):

```
CustomPhysicalRenderer.prototype.Init = function(){
    // Be a good citizen. Call Superclass first
    SiebelAppFacade.CustomPhysicalRenderer.superclass.Init.call(this);
    // Call AttachPMBinding here.
}
```

For example:

```
CustomPhysicalRenderer.prototype.Init = function(){
    SiebelAppFacade.CustomPhysicalRenderer.superclass.Init.call(this);
    this.AttachPMBinding("EndQueryState", EndQueryState);
}
```

Sending an Arbitrary Scope

An *arbitrary scope* is any scope other than the scope that calls the handler. You can configure Siebel Open UI to send to AttachPMBinding any scope that is available in the physical renderer. You can use the BasePhysicalRenderer class to send an arbitrary scope that identifies the handler method that Siebel Open UI must use.

To send an arbitrary scope

- 1 Use a JavaScript editor to open your custom physical renderer.
- 2 Add the following code to send an arbitrary scope as an argument:

```
this.AttachPMBinding ("FocusOnApplet", FocusOnApplet, arbitrary_scope);
```

For example:

```
this.AttachPMBinding ("FocusOnApplet", FocusOnApplet, SiebelAppFacade.S_App);
```

where:

- SiebelAppFacade.S_App is an arbitrary scope because it is not the calling scope that the `this` statement identifies, which Siebel Open UI assumes in BasePR, by default. In this example, the FocusOnApplet handler must exist in the SiebelAppFacade.S_App scope.

Accessing Proxy Objects

If you must write code that accesses a proxy object, then it is strongly recommended that you access this proxy object through a physical renderer. The physical renderer typically exposes the interfaces that allow access to operations on the proxy object. The example in this topic accesses a proxy object for an active control.

To access proxy objects

- 1 Use a JavaScript editor to open your custom physical renderer.
- 2 Add the following code:

```
this.ExecuteMethod("SetActiveControl", control);
```

This example code accesses a proxy object so that Siebel Open UI can modify an active control.

It is recommended that you do not write code that directly accesses a proxy object from a physical renderer. In the following example, Siebel Open UI might remove the GetProxy method from the presentation model, and any code that references GetProxy might fail. It is recommended that you do not use the following code:

```
this.GetProxy().SetActiveControl (control);
```

Modifying Nonapplet Configurations for Siebel CRM Version 8.1.1.10, 8.2.2.3, or Earlier

Siebel Open UI removed the scope argument for calls that it makes to the AttachPMBinding method with nonapplet objects, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4. You can modify your custom code to use this new configuration.

To modify nonapplet configurations for Siebel CRM versions 8.1.1.10, 8.2.2.3, or earlier

- 1 Use a JavaScript editor to open your custom physical renderer.
- 2 Locate the following code:

```
this.GetPM().AttachPMBinding ("FocusOnApplet", FocusOnApplet, {scope: this});
```

In this example, AttachPMBindings uses the scope argument to do a call in Siebel CRM version 8.1.1.10, 8.2.2.3, or earlier.

- 3 Replace the code that you located in [Step 2](#) with the following code:

```
this.AttachPMBinding ("FocusOnApplet", FocusOnApplet);
```

You can use this code starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4.

Customizing Events

This topic includes some examples that describe how to customize Siebel Open UI to use events. For more information about how Siebel Open UI uses events and examples that configure them, see the following topics:

- [How Siebel Open UI Uses the Init Method of the Presentation Model on page 55](#)
- [Life Cycle of a Physical Renderer on page 56](#)
- [Attaching an Event Handler to a Presentation Model on page 78](#)
- [Customizing the Physical Renderer to Bind Events on page 88](#)
- [Customizing the Event Handlers on page 92](#)
- [Siebel CRM Events You Can Use to Customize Siebel Open UI on page 563](#)

Refreshing Custom Events

Siebel Open UI does not come predefined to refresh a custom event. The example in this topic describes how to modify this behavior.

To refresh custom events

- 1 Add the following code:

```
this.s.AddMethod("RefreshHandler", function(x, y, z){
    // Add code here that does processing for RefreshHandler.
});
```

This code adds the RefreshHandler custom event handler.

- 2 Add the following code in the presentation model so that it is aware of the event that the RefreshEventHandler specifies:

```
this.s.AttachEventHandler("Refresh", "RefreshHandler");
```

For more information, see ["AttachEventHandler Method" on page 427](#).

- 3 Add the following code in the physical renderer:

```
controlElement.bind("click", {ctx: this}, function(event){
    event.data.ctx.GetPM().OnControlEvent("Refresh", value1, value2, value3);
});
```

This code binds the physical event to the presentation model. For more information, see ["OnControlEvent Method" on page 432](#).

Overriding Event Handlers

The example in this topic configures Siebel Open UI to override an event handler that the predefined presentation model references.

To override event handlers

- 1 Configure Siebel Open UI to refresh a custom event.

For more information, see ["Customizing Events" on page 118](#).

- 2 Add the following code to your custom presentation model:

```
this.s.AddMethod(SiebelApp.Constants.get("PHYEVENT_INVOKE_CONTROL"),
function(controlName) {
    // Process button click
    return false;
});
```

This code configures Siebel Open UI to return the following value from the event handler. It makes sure this presentation model does not continue processing:

```
false
```

Creating Components

The example in this topic configures Siebel Open UI to attach a local component as the child of a view component, and it uses the property set that Siebel Open UI uses to create this component to specify the name of the module. Siebel Open UI uses this module for the presentation model and the physical renderer.

To create components

- 1 Create the property set. Use the following code:

```
var psInfo = CCFMiscUtil.CreatePropSet();
psInfo.SetProperty(consts.get("SWE_UI_DEF_PM_CTR"), "siebel/custom/customPM");
psInfo.SetProperty(consts.get("SWE_UI_DEF_PR_CTR"), "siebel/custom/customPR");
```

where:

- siebel/custom/customPM is the module name that identifies the siebel/custom/customPM.js presentation model
- siebel/custom/customPR is the module name that identifies the siebel/custom/customPR.js physical renderer

- 2 Create the dependency object. Use the following code:

```
var dependency = {};
dependency.GetName = function(){return "custom_Dependency_object";}
```

This example assumes that it is not necessary that this component reference an applet, so the code limits the scope to a view.

- 3 Call the MakeComponent method. Use the following code:

```
SiebelAppFacade.ComponentMgr.MakeComponent(SiebelApp.S_App.GetActiveView(),
psInfo, dependency);
```

For more information, see ["MakeComponent Method" on page 502](#).

Allowing Users to Interact with Clients During Business Service Calls

The user cannot interact with the client during a synchronous request to a business service until the client receives the reply for this request from the Siebel Server. However, the user can interact with the client while it is waiting for a reply during an asynchronous request. This topic describes how to write JavaScript code so that it sends an asynchronous request that allows the user to continue to use the client without interruption during the call. You use the following code to specify an asynchronous call:

```
async = true or false
```

For example, the following code makes an asynchronous request:

```
async = true
```

To view an example presentation model that includes more than one instance of enabling and disabling an asynchronous call, download the msgbrdcstpsync.js file, and then search this file for the following string:

```
lpsca.async
```

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

For more information, see [“About Synchronous and Asynchronous Requests” on page 74](#).

To allow users to interact with clients during business service calls

- 1 Use a JavaScript editor to open the presentation model that you must modify.
- 2 Locate the ExecuteMethod that calls the business service that you must modify.

Siebel Open UI uses the ExecuteMethod method to call a business service. For more information, see [“ExecuteMethod Method” on page 430](#).

- 3 Add the following code to the AddMethod call that you located in [Step 2](#):

```
var service = SiebelApp.S_App.GetService("service_name");
var inPropSet = SiebelApp.S_App.NewPropertySet ();
// set all the input arguments through inPropSet.SetProperty("property_name",
"property_value")
var outPropSet;
if(service){
    var config = {};
    config.async = true;
    config.scope = this;
    config.cb = function(){
        outPropSet = arguments[2];
        if (outPropSet !== null){
            output_property_set
        }
    }
    service.InvokeMethod ("method_name", inPropSet, config);
}
```

where:

- inPropSet.SetProperty allows you to add input arguments to the business service on the Siebel Server.
- service_name is the name of a business service that Siebel Open UI must call.
- config.async is set to true.
- config.scope = this attaches a scope to the callback function so that you are not required to use var that=this to resolve the scope. For more information, see [“Coding Callback Methods” on page 375](#).
- method_name is the name of a business service method that resides in the business service that you specify in service_name.
- output_property_set is the name of the property set that Siebel Open UI uses to store the output of this asynchronous call.

For example, the following code creates an asynchronous call to create a list of quotes:

```
var service = SiebelApp.S_App.GetService("Create Quote Service");
var inPropSet = SiebelApp.S_App.NewPropertySet ();
var outPropSet;
if(service){
    var config = {};
```

```

config.async = true;
config.scope = this;
config.cb = function(){
    outPropSet = arguments[2];
    if (outPropSet !== null){
        quoteList
    }

    service.invokeMethod ("Create Quote", inPropSet, config);
}

```

where:

- `quoteList` is an output property set that contains a list of quotes that Siebel Open UI gets from the Create Quote business service method.

Managing Files

- [Organizing Files That You Customize on page 122](#)
- [Updating Relative Paths in Files That You Customize on page 125](#)
- [Specifying Dependencies Between Presentation Models or Physical Renderers and Other Files on page 126](#)
- [Specifying the Order That Siebel Open UI Uses to Download Files on page 127](#)

You also configure manifest to manage files. For more information, see [“Configuring Manifests” on page 128](#).

Organizing Files That You Customize

This topic describes how to organize files that you customize. A *predefined file* is a type of file that comes configured ready-to-use with Siebel Open UI. A *custom file* is a predefined file that you modify or a new file that you create. A .png file that you use for your company logo is an example of a custom file. You can customize the following types of files:

- JavaScript files.
- CSS files.
- Image files, such as .jpg or .png files.
- SWT (Siebel Web Template) files.
- HTML files.
- XML files.

Note the following guidelines:

- You must modify any relative paths that your custom file contains. For more information, see [“Updating Relative Paths in Files That You Customize” on page 125](#).
- The folder structures that this topic describes applies to all cached and deployed files.

- Any third-party libraries that you use must reside in a predefined folder or in a custom folder.

CAUTION: You must not modify any files that reside in the folders that [Table 7 on page 124](#) describes. You must make sure that these folders contain only Oracle content, and that your custom folders contain only custom content. This configuration helps to avoid data loss in these folders. If you modify any predefined file, then Siebel Open UI might fail, and it might not be possible to recover from this failure.

To organize files that you customize

- Store all your custom files that reside on the Siebel Server in one of the following folders:

```
ORACLE_HOME\si ebsrvr\WEBMASTER\fi les\language_code\custom
ORACLE_HOME\si ebsrvr\WEBMASTER\i mages\language_code\custom
ORACLE_HOME\si ebsrvr\WEBTEMPL\custom
ORACLE_HOME\si ebsrvr\WEBTEMPL\OUI WEBTEMPL\custom
ORACLE_HOME\si ebsrvr\WEBMASTER\si ebel _bui ld\scri pts\si ebel \custom
```

where:

- *ORACLE_HOME* is the folder where you installed the Siebel Server.
- Store all your custom CSS files and image files that reside on the client in one of the following folders:

```
CLIENT_HOME\PUBLIC\language_code\fi les\custom
CLIENT_HOME\PUBLIC\language_code\i mages\custom
```

where:

- *CLIENT_HOME* is the folder where you installed the client.
- Store all your custom presentation models and physical renderers in the following folder:

```
CLIENT_HOME\PUBLIC\language_code\rel ease_number\scri pts\si ebel \custom
```

Oracle stores predefined presentation models and physical renderers in the following folder. You must not modify any file that resides in this folder:

```
CLIENT_HOME\PUBLIC\language_code\rel ease_number\scri pts\si ebel \
```

- Store all your custom web templates for Siebel Open UI in the following folder:

```
CLIENT_HOME\WEBTEMPL\OUI WEBTEMPL\CUSTOM
```

- Store all your custom web templates for high interactivity and standard interactivity in the following folder:

```
CLIENT_HOME\WEBTEMPL\CUSTOM
```

Where Siebel Open UI Stores Predefined Files in Siebel Open UI Clients

Table 7 describes where Siebel Open UI stores predefined files in the Siebel Open UI client. You must not modify any of these files. Instead, you can copy the file, and then save this copy to one of your custom folders.

Table 7. Where Siebel Open UI Stores Predefined Files in Siebel Open UI Clients

| File Type | Folders Where Siebel Open UI Stores Predefined Files |
|------------------|--|
| JavaScript files | <p>Siebel Open UI stores JavaScript files in the following folders:</p> <pre>CLIENT_HOME\PUBLIC\enu\release_number\scripts CLIENT_HOME\PUBLIC\enu\release_number\scripts\siebel CLIENT_HOME\PUBLIC\enu\release_number\scripts\3rdParty</pre> <p>These folders contain JavaScript files only for predefined Siebel Open UI. You must not modify these files, and you must not store any custom files in these folders. The 3rdParty folder might contain CSS files that the third-party JavaScript files require.</p> |
| CSS files | <p>Siebel Open UI stores CSS files in the following folders:</p> <pre>CLIENT_HOME\PUBLIC\enu\files CLIENT_HOME\PUBLIC\enu\files\3rdParty</pre> <p>These folders contain CSS files only for predefined Siebel Open UI. You must not modify these files, and you must not store any custom files in these folders.</p> |
| Image files | <p>Siebel Open UI stores image files in the following folders:</p> <pre>CLIENT_HOME\PUBLIC\enu\images</pre> <p>These folders contain image files only for predefined Siebel Open UI. You must not modify these files, and you must not store any custom files in this folder. To support color schemes, Siebel Open UI converts the images that Oracle provides from GIF files to PNG files.</p> |

Where Siebel Open UI Stores Files in High-Interactivity and Standard-Interactivity Deployments

Siebel Open UI uses the following folders in a high-interactivity client or standard-interactivity client. This folder structure makes sure existing high-interactivity or standard-interactivity applications and customizations can continue to work simultaneously with Siebel Open UI:

```
ORACLE_HOME\siebsrvr\WEBTEMPL
ORACLE_HOME\siebsrvr\WEBTEMPL\custom
ORACLE_HOME\siebsrvr\WEBTEMPL\OUI WEBTEMPL
ORACLE_HOME\siebsrvr\WEBTEMPL\OUI WEBTEMPL\custom
```

Siebel Open UI uses the following folder only for high-interactivity and standard-interactivity clients:

```
ORACLE_HOME\siebsrvr\WEBTEMPL\custom
```

Siebel Open UI uses the following folders only in the Siebel Open UI client:

```
ORACLE_HOME\siebsrvr\WEBTEMPL\OUIWEBTEMPL
ORACLE_HOME\siebsrvr\WEBTEMPL\OUIWEBTEMPL\custom
```

Siebel Open UI uses the following folder for Siebel Open UI, high-interactivity, and standard-interactivity clients:

```
ORACLE_HOME\siebsrvr\WEBTEMPL
```

If more than one folder contains web templates with the same name, then Siebel Open UI uses the first file that it encounters according to the following search order:

- For Siebel Open UI clients:
 - `ORACLE_HOME\siebsrvr\WEBTEMPL\OUIWEBTEMPL\custom`
 - `ORACLE_HOME\siebsrvr\WEBTEMPL\OUIWEBTEMPL`
 - `ORACLE_HOME\siebsrvr\WEBTEMPL`
- For high-interactivity and standard-interactivity clients:
 - `ORACLE_HOME\siebsrvr\WEBTEMPL\custom`
 - `ORACLE_HOME\siebsrvr\WEBTEMPL`

Updating Relative Paths in Files That You Customize

If you customize a file, and if you save this custom file in a custom folder, then you must modify any relative paths that this file references. For example, if you copy the rules from a predefined .css file into a custom .css file, then you must modify the relative paths that your custom .css file references so that they reference the correct file. For an example of this configuration, see [“Customizing the Logo” on page 146](#).

To update relative paths in files that you customize

- 1 Create a custom file.
For more information about custom files, see [“Organizing Files That You Customize” on page 122](#).
- 2 Search your custom file for any relative paths.
For example, `images/` in the following code is an example of a relative path:

```
src=images/ebus.gif
```
- 3 Modify the relative path so that it can correctly locate the file that it references.
For example:

```
src=CLIENT_HOME/eappweb/PUBLIC/enu/images/ebus.gif
```
- 4 Do [Step 2](#) and [Step 3](#) for every relative path that your custom file contains.

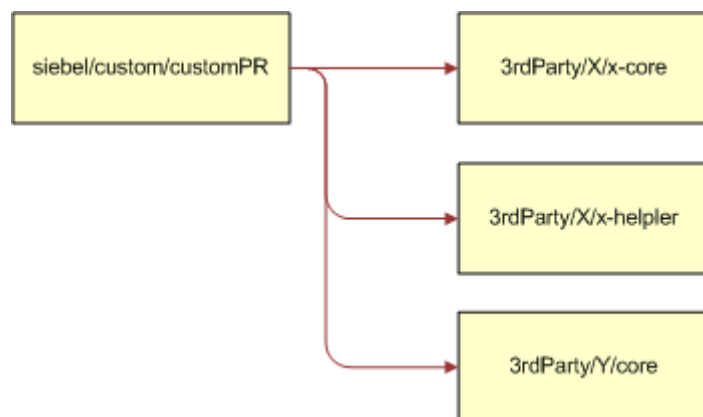
Specifying Dependencies Between Presentation Models or Physical Renderers and Other Files

A presentation model or physical renderer sometimes includes a *module dependency*, which is a relationship that occurs when this presentation model or physical renderer depends on another file. The Define method recognizes each of these items as a *JavaScript code module*, which is an object that the module_name argument identifies as depending on other modules to run correctly. You specify the module_name argument when you use the Define method to identify the JavaScript files that Siebel Open UI must download for a presentation model or physical renderer. For more information, see [“Define Method” on page 506](#).

Consider the following example that uses the customPR.js file to define the physical renderer. This renderer depends on plug-in X and plug-in Y, and it uses the following directory structure:

- 3rdParty
 - X
 - x-core.js
 - x-helper.js
 - Y
 - core.js
- siebel
 - custom
 - customPR.js

In this example, the following logical dependencies exist between the customPR.js file and the x-core.js file, x-helper.js file, and the customPR.js file:



Siebel Open UI then uses the following logic at run-time for this example:

- 1 The user navigates to a view that includes an applet that uses the customPR physical renderer.

- 2 The Siebel Server sends a reply to the client that includes information about the property set and the physical layout.
- 3 The view processes the information that the Siebel Server sends in [Step 2](#), and then determines that it must use siebel/custom/customPR.js to render the applet.
- 4 The RequireJS script loader uses the customPR.js file name to identify siebel/custom/customPR as the module name, and then sends a request to the Siebel Server for this module.
- 5 If Siebel Open UI already loaded this module, then it returns the module object to the client and proceeds to [Step 7](#).
- 6 If Siebel Open UI has not already loaded this module, then it does the following work:
 - a Sends a request to the web server for the siebel/custom/customPR.js file.
 - b If dependencies exist, then Siebel Open UI sends a request for these dependent modules, and then runs the modules in the browser.
 - c Siebel Open UI runs the script for the siebel/custom/customPR.js file in the browser.
- 7 Siebel Open UI uses the module object to create a new instance of the presentation model and the physical renderer.

To help manage your customizations, it is strongly recommended that you use a module name that is similar to the relative location of the file name. You use the manifest administration screens to specify the manifest for these dependencies.

To specify dependencies between presentation models or physical renderers and other files

- Use the list_of_dependencies argument when you use the Define method in your presentation model or physical renderer.

For an example that uses the list_of_dependencies argument, see [“Setting Up the Physical Renderer” on page 84](#). For more information, see [“Define Method” on page 506](#).

- If file dependencies require that you configure Siebel Open UI to download files in a specific order, then do [“Specifying the Order That Siebel Open UI Uses to Download Files” on page 127](#).

Specifying the Order That Siebel Open UI Uses to Download Files

In some situations, you must configure Siebel Open UI to use a specific order when it downloads and runs JavaScript files in the client.

To specify the order that Siebel Open UI uses to download files

- Use the following code:

```
define("siebel/custom/module_name", ["order! dependent_module1", "
order! dependent_module2", "order! dependent_modulen"], function() {
    . . .
})
```

```

    . . .
    . . .
    return "name_space.module_name";
  });

```

where:

- `order!` is a prefix that specifies to download and run dependent modules in the order that you specify them. You add this prefix immediately before the module name. Siebel Open UI downloads and runs these modules in the order that you specify them.
- `dependent_module` specifies the file name that Siebel Open UI must load and the relative path where this file resides.

For example, assume the siebel/phyrenderer file depends on the following file:

```
3rdParty/X/x-core.js
```

In this example, Siebel Open UI must download and run the 3rdParty/X/x-core.js file before it downloads and runs the siebel/phyrenderer file. You can use the following code to specify this order:

```

define("siebel/custom/name-module", ["order! 3rdParty/X/x-core", "order! siebel /
phyrenderer"], function(){
    . . .
    . . .
    . . .
    return "name_space.module_name";
  });

```

It is recommended that you use the `order!` prefix only if necessary. For information about file dependencies, see ["Specifying Dependencies Between Presentation Models or Physical Renderers and Other Files" on page 126](#).

Configuring Manifests

This topic describes how to configure Siebel Open UI manifests. It includes the following topics:

- ["Overview of Configuring Manifests" on page 128](#)
- ["Configuring Custom Manifests" on page 132](#)
- ["Adding Custom Manifest Expressions" on page 143](#)
- ["Adding JavaScript Files to Manifest Administrative Screens" on page 144](#)

Overview of Configuring Manifests

A *manifest* is a set of instructions that Siebel Open UI uses to identify the JavaScript files that it must download from the Siebel Server to the client so that it can render screens, views, and applets. For an overview of how Siebel Open UI uses this manifest, see ["Example of How Siebel Open UI Renders a View or Applet" on page 41](#).

Siebel CRM versions 8.1.1.9 and 8.1.1.10 use an XML manifest file to identify these JavaScript files in the following situations:

- When Siebel Open UI initializes the Siebel application. Siebel Open UI does this download only for one Siebel application at a time.
- The first time Siebel Open UI must display an applet in a Siebel application.

Starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4, Siebel Open UI replaces the XML manifest file with manifest data that it stores in the Siebel Database. You cannot modify this predefined manifest data, but you can use the Manifest Administration screen in the client to configure the manifest data that your customization requires. For information about using a utility that migrates your custom manifest configurations from Siebel CRM version 8.1.1.9 or 8.1.1.10 to version 8.1.1.11 or 8.2.2.4, see the topic about migrating the Siebel Open UI manifest file in *Siebel Database Upgrade Guide*.

Example of How Siebel Open UI Identifies the JavaScript Files It Must Download

Figure 29 describes an example of how Siebel Open UI uses the manifest to identify the JavaScript file it must download so that it can use the presentation model for the SIS Account List Applet. The manifest maps the siebel/custom/recyclebinmodel.js file to the presentation model that it uses to display this applet. For details about this example, see “Creating the Presentation Model” on page 62 and “Configuring the Manifest for the Recycle Bin Example” on page 94.

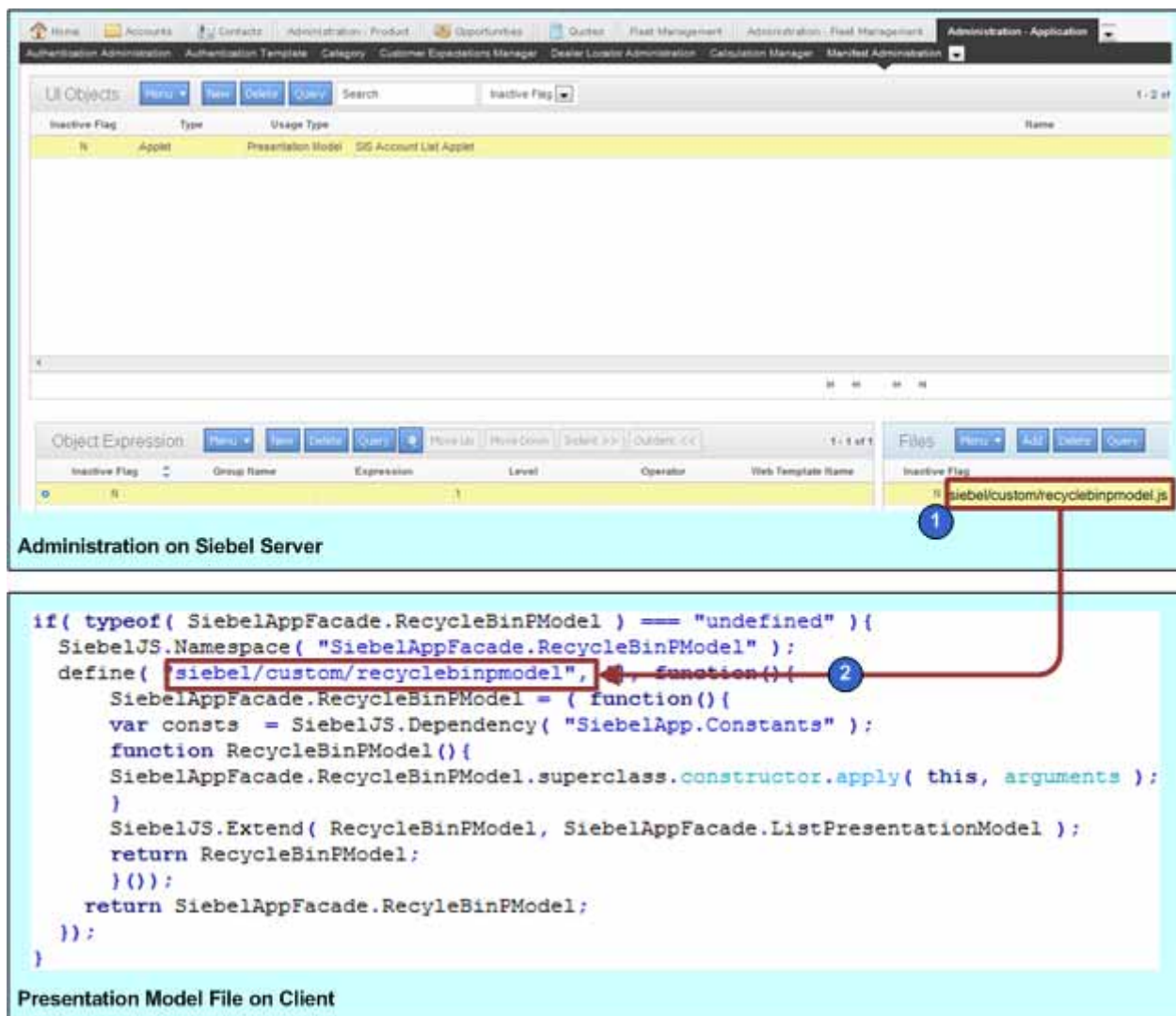


Figure 29. Example of How Siebel Open UI Identifies the JavaScript Files It Must Download

Explanation of Callouts

The example manifest administration includes the following items:

- 1 The Files list specifies the siebel/custom/recyclebinmodel.js file.

- The presentation model specifies siebel/custom/recyclebinmodel when it calls the define method.

Example of a Completed Manifest Administration

Figure 30 includes an example of a completed manifest administration that configures Siebel Open UI to download JavaScript files for the Contact List Applet. For information about how to configure this example, see [“Configuring Custom Manifests” on page 132](#).

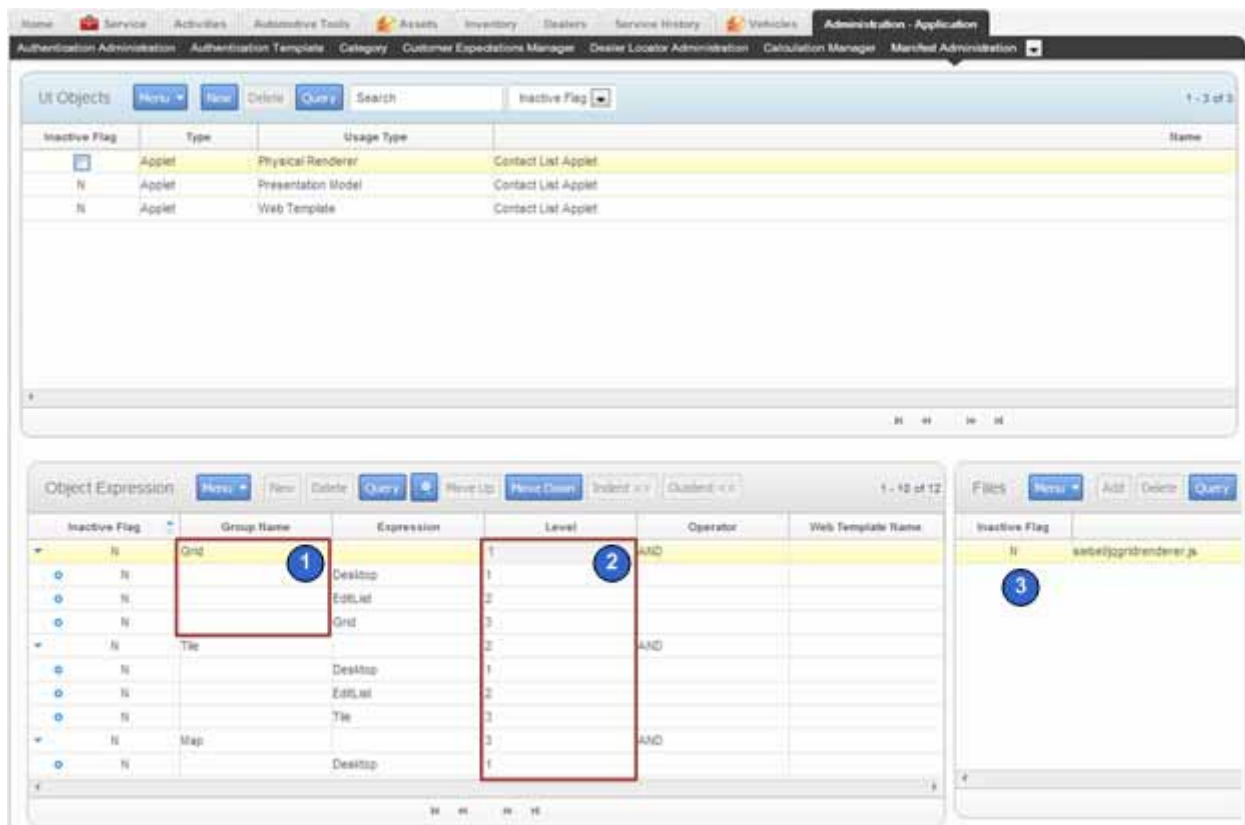


Figure 30. Example Manifest Administration

Explanation of Callouts

The example manifest administration includes the following items:

- The Grid group uses the AND operator to group three expressions into the following group expression:

Desktop AND EditList AND Grid

A *group expression* is a type of expression that Siebel Open UI uses to arrange subexpressions into a group in the Object Expression list.

2 Siebel Open UI uses the Level field to determine the order it uses to evaluate expressions. It uses the following sequence:

- a It uses the Level field to determine the order it uses to evaluate group expressions. In this example, it uses the following sequence:
 - Evaluates the Grid group first.
 - Evaluates the Tile group next.
 - Evaluates the Map group last.
- b It uses the Level field within a group to determine the order it uses to evaluate each *subexpression*, which is a type of expression that Siebel Open UI displays as part of a group in the Object Expressions list. It displays each subexpression in an indented position below the group expression. In this example, it uses the following sequence to evaluate subexpressions that reside in the Grid group:
 - Evaluates the Desktop expression first.
 - Evaluates the EditList expression next.
 - Evaluates the Grid expression last.

In this example, Siebel Open UI evaluates all the expressions that reside in the Grid group, and then does one of the following according to the result of this evaluation:

- **All expressions that reside in the Grid group evaluate to true.** Siebel Open UI downloads the file that the Files list specifies.
 - **Any expression that resides in the Grid group evaluates to false.** Siebel Open UI discards the entire Grid group, and then evaluates the Tile group.
- 3 Siebel Open UI uses the Files list to identify the files it must download. In this example, it does the following evaluation:
- If the platform is a desktop, and if the mode is EditList, and if the user chooses Grid, then it downloads the siebel/jqgridrenderer.js file.
 - If the platform is a desktop, and if the mode is EditList, and if the user chooses Tile, then it downloads the siebel/Tilescrollcontainer.js file.

To view an example that allows the user to choose Grid or Tile, see [“Allowing Users to Change the Applet Visualization” on page 182](#).

Configuring Custom Manifests

This topic describes how to configure the example described in [“Example of a Completed Manifest Administration” on page 131](#). For other examples that configure the manifest to download objects for:

- Web templates and modified applet modes, see [“Allowing Users to Change the Applet Visualization” on page 182](#), and [“Customizing Tiles for List Applets” on page 306](#).
- Different web templates, physical renderers, and presentation models depending on the applet, see [“Displaying Applets Differently According to the Applet Mode” on page 190](#).

- Phones, tablets, or desktops, see [“Customizing the Number of Columns in Mobile Applets” on page 292](#).
- Desktops or mobile platforms, see [“Customizing Mobile Lists” on page 299](#).
- A physical renderer that renders tiles on a desktop, see [“Customizing Tiles for List Applets” on page 306](#).
- The physical renderer and the presentation model, see [“Configuring the Manifest for the Recycle Bin Example” on page 94](#).
- Only for the physical renderer, see [“Customizing Tiles for Mobile Lists” on page 312](#).

To configure custom manifests

- 1 Make sure your custom presentation model or physical renderer uses the Define method:
 - a Use a JavaScript editor to open your custom presentation model or physical renderer.
 - b In the section where you configure Siebel Open UI to do the setup, make sure you use the Define method to identify the presentation model file or the physical renderer file.

For an example that does this setup, see [“Example of How Siebel Open UI Identifies the JavaScript Files It Must Download” on page 130](#).

- 2 Configure the manifest files:
 - a Log in to the Siebel client with administrative privileges.
 - b Navigate to the Administration - Application screen, and then the Manifest Files view.
 - c Verify that the Manifest Files view includes the files that Siebel Open UI must download for your custom deployment.

For this example, verify that the Manifest Files view includes the following files:

```

siebel /l i stappl et. j s
siebel /j qqri drenderer. j s

```

If the Manifest Files view does not include these files, then add them now. For more information, see [“Adding JavaScript Files to Manifest Administrative Screens” on page 144](#).

- 3 Configure the UI object:
 - a Navigate to the Administration - Application screen, and then the Manifest Administration view.
 - b In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Contact List Applet |

For information, see [“Fields of the UI Objects List” on page 137](#).

4 Configure the Grid group:

For information about how to configure a group, see [“Adding Group Expressions” on page 140](#).

- a** In the Object Expression list, add the following subexpression.

| Field | Value |
|-------------------|--------------|
| Group Name | Leave empty. |
| Expression | Desktop |
| Level | 1 |
| Operator | Leave empty. |
| Web Template Name | Leave empty. |

For information, see [“Fields of the Object Expression List” on page 139](#).

- b** Add another subexpression.

| Field | Value |
|-------------------|--------------|
| Group Name | Leave empty. |
| Expression | EditList |
| Level | 2 |
| Operator | Leave empty. |
| Web Template Name | Leave empty. |

- c** Add another subexpression.

| Field | Value |
|-------------------|--------------|
| Group Name | Leave empty. |
| Expression | Grid |
| Level | 3 |
| Operator | Leave empty. |
| Web Template Name | Leave empty. |

- d** Add the following group expression.

| Field | Value |
|------------|--------------|
| Group Name | Leave empty. |
| Expression | Grid |
| Level | 1 |

| Field | Value |
|-------------------|--------------|
| Operator | Leave empty. |
| Web Template Name | Leave empty. |

- e** Use the Move Up and Move Down buttons to arrange the subexpressions in ascending numeric order according to the value in the Level field. Make sure the Object Expression list displays all subexpressions below the group expression.
- f** Use the Indent and Outdent buttons so that Siebel Open UI displays the subexpressions below and indented from the group expression. The tree in the Inactive Flag field displays this indentation.
- g** In the UI Objects list, query the Name property for the name of the UI object that you are configuring. This query refreshes the Manifest Administration screen so that you can edit the Group Name and Operator fields of the group expression.
- h** In the Object Expressions list, expand the tree that Siebel Open UI displays in the Inactive Flag field.
- i** Set the following fields of the group expression.

| Field | Value |
|------------|-------|
| Group Name | Grid |
| Operator | AND |

- 5** Specify the files that Siebel Open UI must download for the Grid group:
 - a** Make sure the Grid group expression is chosen in the Object Expression list.
 - b** In the Files list, click Add.
 - c** In the Files dialog box, click Query.
 - d** In the Name field, enter the path and file name of the file.

For example, enter the following value:

siebel /jqgridrender.js

- e** Click Go.

If the Files dialog box does not return the file that your deployment requires, then you must use the Manifest Files view to add this file before you can specify it in the Files list. For more information, see [“Adding JavaScript Files to Manifest Administrative Screens” on page 144](#).

- f** Click OK.

- 6** Configure the Tile group:

- a** Repeat [Step 4](#), with the following differences:
 - ☐ For the group expression, set the Group Name field to Tile and the Level field to 2.
 - ☐ For the last subexpression, set the Expression field to Tile.

- b** Repeat [Step 5](#), except add the following file:

`siebel/tilescrollrenderer.js`

7 Configure the Map group:

- a** Repeat [Step 4](#), with the following differences:

- ☐ For the group expression, set the Group Name field to Map and the Level field to 3.
- ☐ Add only one subexpression with the Expression field set to Map.

- b** Repeat [Step 5](#), except add the following file:

`siebel/custom/siebelmaprenderer.js`

- 8** In the Object Expression list, use the Move Up, Move Down, Indent, and Outdent buttons until the Object Expression list resembles the configuration in [Figure 30](#).

Fields of the UI Objects List

Table 8 describes the fields of the UI Objects list.

Table 8. Fields of the UI Objects List

| Field | Description |
|---------------|--|
| Inactive Flag | <p>Set to one of the following values:</p> <ul style="list-style-type: none"> ■ Y. Make the object inactive. Make sure you set the Inactive Flag to Y for any custom object that your deployment does not require. ■ N. Make the object active. Make sure you set the Inactive Flag to N for any custom object that your deployment requires. <p>The Inactive Flag allows you to configure more than one manifest. You can activate or deactivate each of these configurations during development. You can set the Inactive Flag in the same way for each object that the Manifest Administration view displays.</p> |
| Type | <p>Choose one of the following values to specify the type of Siebel CRM object that you are customizing:</p> <ul style="list-style-type: none"> ■ Application ■ View ■ Applet ■ Navigation ■ Toolbar ■ Menu ■ Control <p>For more information, see “How Siebel Open UI Chooses Files If Your Custom Manifest Matches a Predefined Manifest” on page 141.</p> |

Table 8. Fields of the UI Objects List

| Field | Description |
|------------|---|
| Usage Type | <p>Specify how Siebel Open UI must download files. Choose one of the following values:</p> <ul style="list-style-type: none"> ■ Common. Siebel Open UI downloads the files when it initializes the Siebel application. Siebel Call Center is an example of a Siebel application. ■ Theme. Siebel Open UI downloads only the files it requires to support a theme that you customize. ■ Presentation Model. Siebel Open UI downloads the files that your custom presentation model requires. ■ Physical Renderer. Siebel Open UI downloads the files that your custom physical renderer requires. ■ Web Template. Siebel Open UI downloads files according to the Name property of the web template file. You specify this web template file in the web template in Siebel Tools. For more information, see “Identifying the Web Template File Name” on page 142. <p>For more information, see “How Siebel Open UI Chooses Files If Your Custom Manifest Matches a Predefined Manifest” on page 141.</p> |
| Name | Enter the name of your custom object. For example, if you set the Type to Applet, then you must specify the value that Siebel Tools displays in the Name property of the applet. |

Fields of the Object Expression List

[Table 9](#) describes the fields of the Object Expression list. You can configure a simple expression, or you can configure a complex expression that includes AND or OR operators, and that can include nested levels. For an example that includes complex expressions, see [“Configuring Custom Manifests” on page 132](#).

Table 9. Fields of the Object Expression List

| Field | Description |
|------------|---|
| Group Name | <p>If the record that you are adding to the Object Expressions list is part of a group of two or more expressions, and if this record is the group expression, then enter a value in the Group Name field and leave the Expression field empty.</p> <p>The Object Expressions list is a hierarchical list. You can use it to specify complex expressions that you enter as more than one record in this list.</p> <p>You must add more than one record and indent at least one of them before you can enter a group name. For information about how to do this work, see “Adding Group Expressions” on page 140.</p> |
| Expression | <p>If the record that you are adding to the Object Expressions list is:</p> <ul style="list-style-type: none"> ■ Not a group expression. Set a value in the Expression field and leave the Group Name field empty. ■ A group expression. Leave the Expression field empty and enter a value in the Group Name field. <p>If the Expression list does not include the expression that your deployment requires, then you must add a custom expression. For more information, see “Adding Custom Manifest Expressions” on page 143.</p> |
| Level | <p>Enter a number to determine the order that Siebel Open UI uses to evaluate expressions that the Object Expression list contains. Siebel Open UI evaluates these expressions in ascending, numeric order according to the values that the Level field contains. If the Type field in the UI Objects list:</p> <ul style="list-style-type: none"> ■ Is Application, then Siebel Open UI evaluates every expression. It downloads each file that the Files list specifies for each expression that it evaluates to true. ■ Is not Application, and if Siebel Open UI evaluates an expression to true, then it does the following: <ul style="list-style-type: none"> ■ Downloads the file that the Files list specifies for this expression ■ Does not process any expression that exists further down in the order ■ Does not download any other files |

Table 9. Fields of the Object Expression List

| Field | Description |
|-------------------|--|
| Operator | <p>If the record that you are adding to the Object Expressions list is a group expression, then you must specify the logical operator that Siebel Open UI uses to combine the subexpressions that the group contains. You can use one of the following values:</p> <ul style="list-style-type: none"> ■ AND. Specifies to combine subexpressions. If you specify AND, then Siebel Open UI downloads files only if it evaluates every subexpression in the group to true. ■ OR. Specifies to consider individually each subexpression that resides in the group. If you specify OR, then Siebel Open UI downloads files according to the first subexpression that it evaluates to true. <p>If the record that you are adding to the Object Expressions list is not a group expression, or if it does not reside at the top of the hierarchy, then leave the Operator field empty.</p> |
| Web Template Name | <p>If you set the Usage Type field in the UI Objects list to Web Template, then you must specify the name of the Siebel CRM web template file in the Web Template Name field. To identify this file name, see “Identifying the Web Template File Name” on page 142.</p> |

Adding Group Expressions

You must use the sequence that this topic describes when you add a group expression. For an example that uses this sequence, see [“Configuring Custom Manifests” on page 132](#). For more information about group expressions and subexpressions, see [“Example of a Completed Manifest Administration” on page 131](#).

To add group expressions

- 1 Navigate to the Administration - Application screen, and then the Manifest Administration view.
- 2 In the UI Objects list, locate the UI object that you must modify.
- 3 In the Object Expression list, add the subexpressions.
- 4 Add the group expression. Leave the Group and Operator fields empty.
- 5 Use the Move Up and Move Down buttons to arrange the subexpressions in ascending numeric order according to the value in the Level field. Make sure the Object Expression list displays all subexpressions below the group expression.
- 6 Use the Indent and Outdent buttons so that Siebel Open UI displays the subexpressions below and indented from the group expression. The tree in the Inactive Flag field displays this indentation.
- 7 In the UI Objects list, query the Name property for the name of the UI object that you are configuring. This query refreshes the Manifest Administration screen so that you can edit the Group Name and Operator fields of the group expression.

- 8 In the Object Expressions list, expand the tree that Siebel Open UI displays in the Inactive Flag field.
- 9 Set the values for the Group Name field and the Operator field of the group expression.

How Siebel Open UI Chooses Files If Your Custom Manifest Matches a Predefined Manifest

If the values that you specify in the Type, Usage Type, and Name fields of the UI Objects list are identical to the values that a predefined UI object specifies, then Siebel Open UI uses your custom manifest. For example, Siebel Open UI comes predefined with a UI Object record with the Type set to Applet, the Usage Type set to Physical Renderer, and the Name set to Contact List Applet. To override this configuration, you must do the following work:

- Create a new record in the UI Objects list that contains the same values in the Type, Usage Type, and Name fields that the predefined record contains.
- Add a new record in the Object Expression list that evaluates to true.
- Add a new record in the Files list for the object expression that evaluates to true.

The only exception to this rule occurs in the following situation:

- You set the Type to Application.
- You set the Usage Type to Common.
- A winning expression exists in your customization. A *winning expression* is an expression that Siebel Open UI evaluates to true, and that Siebel Open UI then uses to identify the files it must download according to the configuration that the Manifest Administration view specifies.

In this situation, Siebel Open UI downloads the files that:

- The predefined manifest configuration specifies
- The winning expression of your custom manifest configuration specifies

Table 10 describes how Siebel Open UI chooses files if your manifest configuration matches the predefined manifest configuration for a UI object. The Configuration column describes values that the UI Objects list of the Manifest Administration screen contains.

Table 10. How Siebel Open UI Chooses Files If Your Custom Manifest Matches the Predefined Manifest

| Configuration | Predefined Configuration Exists | Custom Configuration Exists | Result |
|--|---------------------------------|-----------------------------|---|
| Type is Application and Usage Type is Common | Yes | No | Siebel Open UI downloads files according to the winning predefined expressions. |
| Type is Application and Usage Type is Common | Yes | Yes | Siebel Open UI downloads files according to the winning predefined expression and the winning custom expressions. |
| Usage Type is not Common | Yes | No | Siebel Open UI downloads files according to the first predefined expression that it evaluates to true. If more than one expression exists, then it uses the level to determine the sequence it uses to evaluate these expressions. |
| Usage Type is not Common | Yes | Yes | Siebel Open UI downloads files according to the first custom expression that it evaluates to true. If more than one expression exists, then it uses the level to determine the sequence it uses to evaluate these expressions. If Siebel Open UI does not evaluate any custom expression to true, then it uses a predefined expression for this object. |

Identifying the Web Template File Name

This topic describes how to identify the file name that a web template uses.

To identify the web template file name

- 1 Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- 2 In the Object Explorer, click Web Template.
- 3 In the Web Templates list, locate the object definition for the web template.
For example, if you entered Applet Form Grid Layout in the Name field in the UI Objects list, then query the Name property in the Web Templates list for Applet Form Grid Layout.
- 4 In the Object Explorer, expand the Web Template tree, and then click Web Template File.
- 5 In the Web Template Files list, note the value that Siebel Tools displays in the Filename property.
For example, Siebel Open UI uses the CCAppletFormGridLayout.swt file for the Applet Form Grid Layout web template.

Adding Custom Manifest Expressions

This topic describes how to add a custom manifest expression.

To add custom manifest expressions

- 1 Log in to the Siebel client with administrative privileges.
- 2 Navigate to the Administration - Application screen, and then the Manifest Expressions view.
- 3 In the Expressions list, add the following expression.

| Field | Value |
|------------|--|
| Name | <p>Enter text that describes the expression. For example, enter the following value:</p> <p>Desktop</p> <p>Siebel Open UI uses this value as an abbreviation for the expression that it displays in the Expression field in the Object Expression list in the Manifest Administration screen. It uses this abbreviation only to improve readability of the Object Expression list.</p> |
| Expression | <p>Enter an expression. For example:</p> <p>GetProfileAttr("Platform Name") = 'Desktop'</p> <p>Siebel Open UI uses this value when it evaluates expressions that reside in the Object Expression list. For more information, see "GetProfileAttr Method" on page 483.</p> |

Adding JavaScript Files to Manifest Administrative Screens

This topic describes how to add a JavaScript file to the manifest administrative screens.

To add JavaScript files to manifest administrative screens

- 1 Log in to the Siebel client with administrative privileges.
- 2 Navigate to the Administration - Application screen, and then the Manifest Files view.
- 3 In the Files list, add a new record for each JavaScript file that you must add.

Make sure you include the path. For example, to add the mycustomrender.js file, you add the following value:

```
custom/mycustomrender.js
```

You can now add this file in the Files list in the Manifest Administration view. For more information about how to do this, see [Step 5 on page 135](#).

6

Customizing Styles, Applets, and Fields

This chapter describes how to customize styles, applets, fields. It includes the following topics:

- [Customizing Client Logo, Background, and Style on page 145](#)
- [Customizing Applets on page 159](#)
- [Customizing Fields on page 203](#)

Customizing Client Logo, Background, and Style

This topic describes how to customize client logo, background, and style. It includes the following information:

- [Customizing the Logo on page 146](#)
- [Customizing the Background Image on page 147](#)
- [Customizing Browser Tab Labels on page 150](#)
- [Using Cascading Style Sheets to Modify Position, Dimension, and Text Attributes of an Object on page 151](#)
- [Using Cascading Style Sheet Classes to Modify HTML Elements on page 153](#)
- [Customizing the Sequence That Siebel Open UI Uses to Load Cascading Style Sheets on page 156](#)

You can make these modifications in the client at run time. You can then copy them into CSS files on the Siebel Server, and then deploy them to all users.

Customizing the Logo

Starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4, Siebel Open UI defines the logo that it displays in the client in CSS files instead of coding the logo in a web template. It uses the following predefined code to display the logo in the Gray and Tangerine themes:

```
.siebui -l ogo {  
  margin-top: 8px;  
  margin-left: 2px;  
  background-image: url ('../images/ebus.gif');  
  width: 106px;  
  height: 16px;  
}
```

You can configure Siebel Open UI to override this code, or you can create your own custom theme so that you can display a custom logo. You can configure Siebel Open UI to display a separate logo in each theme. For more information about overriding an existing theme, or adding a new theme, see [“Customizing Transitions, Themes, Styles, and Colors” on page 322](#) and Open UI Deployment Guide (Article ID 1499842.1) on My Oracle Support.

To customize the logo

- 1 Create a JPG file that includes your custom logo.

For example, my-logo.jpg.

- 2 Copy the file you created in [Step 1](#) to the following folders:

```
ORACLE_HOME\sies\si ebsrvr\WEBMASTER\images\language_code\custom  
CLIENT_HOME\eappweb\PUBLIC\language_code\images\custom
```

- 3 Use an editor to open your custom CSS file that resides in one of the following folders:

```
ORACLE_HOME\sies\si ebsrvr\WEBMASTER\bui ld_number\scripts\siebel \custom  
CLIENT_HOME\eappweb\PUBLIC\language_code\bui ld_number\scripts\siebel \custom
```

For example, open the my-style.css file.

- 4 Add the following code:

```
.siebui -l ogo {  
  background-image: url ('../images/custom/my-l ogo.jpg')  
}
```

- 5 Use a JavaScript editor to open the theme.js file that resides in the following folder:

```
scripts\siebel \custom\
```

If you are customizing a Siebel Mobile application, then open the mobiletheme.js instead of the theme.js file.

- 6 Add the following code:

```
Siebel App. ThemeManager. addResource (  
  "GRAY_TAB",  
  {
```

```

        css: {
            "sbe-theme": "files/custom/my-style.css"
        }
    }
};

```

If you are customizing a Siebel Mobile application, then replace GRAY_TAB with SBL-MOBILE in this code.

7 (Optional) Modify the logo attributes, as necessary:

- a** Use an editor to open your custom structure CSS file.

For example, my-structure.css.

- b** Add your custom code.

Siebel Open UI uses the following predefined code to specify the logo attributes:

```

.siebui -logo {
    float: left;
    white-space: nowrap;
    width: 50px;
    height: 50px;
}

```

You can modify each of these attributes, as necessary. For example, you can modify the following width and height attributes to decrease the width and height of the logo to accommodate your custom logo image:

```

.siebui -logo {
    width: 25px;
    height: 25px;
}

```

- c** Add the following code to the JavaScript file that you edited in [Step 5](#):

```

SiebelApp.ThemeManager.addResource (
    "GRAY_TAB",
    {
        css: {
            "sbe-theme": "files/custom/my-structure.css"
        }
    }
);

```

If you are customizing a Siebel Mobile application, then replace GRAY_TAB with SBL-MOBILE in this code.

8 Log in to the client and verify that Siebel Open UI replaces the Oracle logo.

Customizing the Background Image

This topic describes how to add a background image in the client.

To customize the background image

- 1 Log in to the Siebel Open UI client using Google Chrome.
- 2 In the application-level menu, right-click the empty area to the right of the Help menu, and then click Inspect Element.
- 3 Enter the following text in the search window:

`_sweclient`

For assistance, see the screen capture in [Step 6 on page 154](#).

- 4 In the main window of the Developer Tools window, click the first child div element of the `_sweclient` element. You click the element that begins with the following code:
- 5 Collapse the Styles pane that the Developer Tools displays to the right of the main window, and then expand the Metrics pane.
- 6 Add the padding:

`1418 x 325`

- a Click in the padding area of the Metrics pane.
- b Double-click the dash that the padding area displays immediately to the left of the following text:

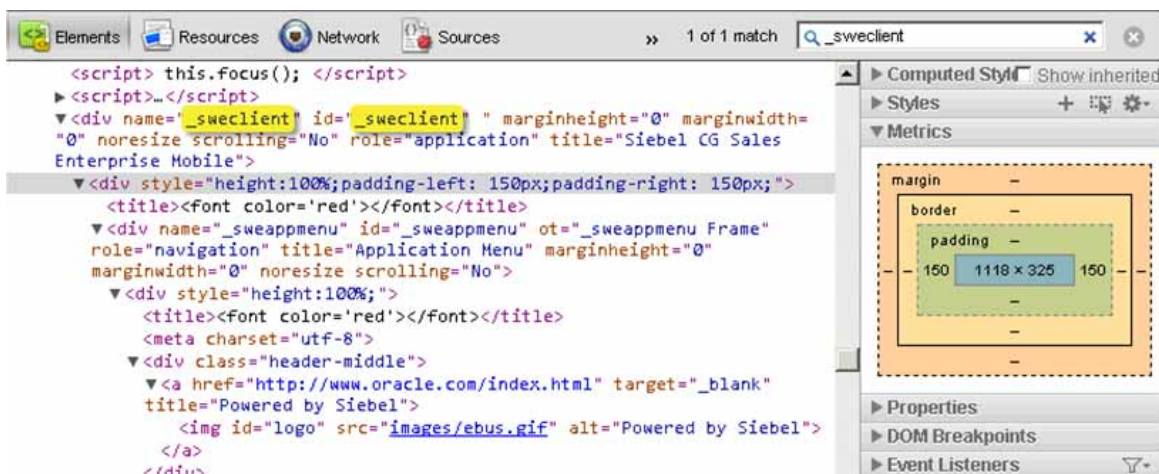
`1418 x 325`

- c Enter 150.
- d Double-click the dash that the padding area displays immediately to the right of the following text:

`1418 x 325`

- e Enter 150.

Use the following screen capture for assistance:

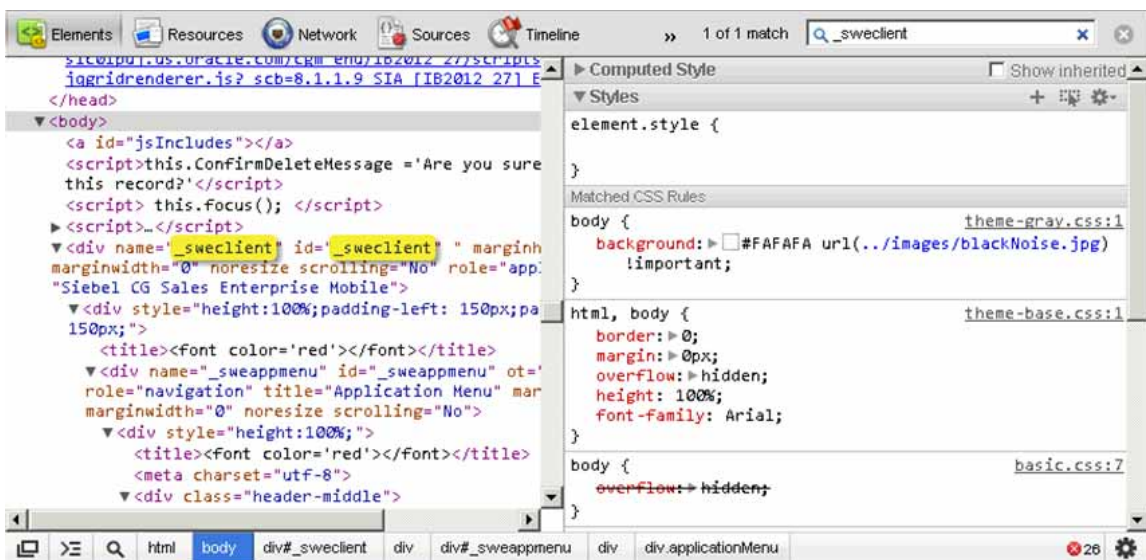


- 7 Add the background image:

- a In the main window of the Developer Tools window, scroll up, and then click the first body tag that occurs above the following element that you located in [Step 3](#):

```
div name="_sweclient"
```
- b Collapse the Metrics pane that the Developer Tools displays to the right of the main window, and then expand the Styles pane.
- c In the Matched CSS Rules section of the Styles pane, modify the background attribute to use the blackNoise.jpg background image.

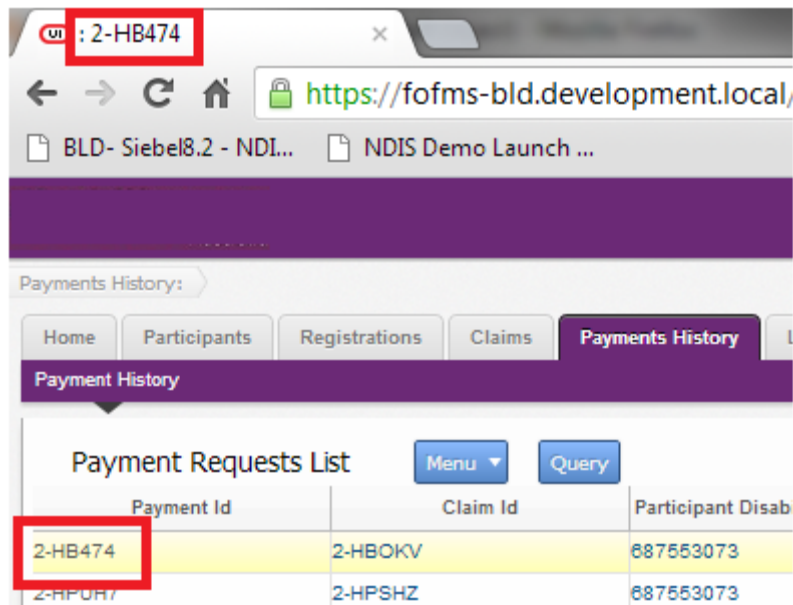
Use the following screen capture for assistance:



- d Note that Siebel Open UI modifies the background image immediately after you specify the new .jpg file.

Customizing Browser Tab Labels

Siebel Open UI uses the view Title that you define in Siebel Tools to set the Browser tab label. If this Title is not defined, then Siebel Open UI displays the Id of the current record as the label. For example, it might display 2-HB474 as the Browser tab label:



This topic describes how to customize Siebel Open UI so that it displays the view Title as the label.

To customize Browser tab labels

- 1 Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- 2 In the Object Explorer, click View.
- 3 In the Views list, query the Name property for the view that you must modify.
- 4 Enter a value in the Title property or the Title - String Override property.

For more information about setting these properties, see *Configuring Siebel Business Applications*.

- 5 Compile your modifications.
- 6 Test your modifications:
 - a Log in to the Siebel Open UI client.
 - b Navigate to the view you located in [Step 3](#).
 - c Verify that the Browser tab label displays the value you entered in [Step 4](#).

Using JavaScript to Customize the Browser Tab Label

This topic describes how to use JavaScript instead of Siebel Tools to customize the browser tab label.

To use JavaScript to customize the browser tab label

- 1 Locate the following code:

```
Siebel App. S_App. GetActiveView(). GetTitle()
```

Siebel Open UI uses this code to get the browser tab label from the SRF. GetTitle returns the value of the Title property that you define in Siebel Tools. You can configure Siebel Open UI to override this value.

- 2 Override the value that Siebel Open UI gets from the SRF. Replace the code you located in [Step 3](#) with the following code:

```
Siebel App. S_App. GetActiveView(). GetTitle()  
"label"
```

where:

- *label* is a text string. Siebel Open UI displays this string as the Browser tab label.

For example, the following code displays Registration Details as the Browser tab label:

```
Siebel App. S_App. GetActiveView(). GetTitle()  
"Registration Details"
```

- 3 Test your modifications.

Using Cascading Style Sheets to Modify Position, Dimension, and Text Attributes of an Object

The example in this topic describes how to modify the cascading style sheet. You move the Predefined Query (PDQ) to a different location and you modify the text color of the Predefined Query.

To use cascading style sheets to modify position, dimension, and text attributes of an object

- 1 Log in to the Siebel Open UI client using Google Chrome.
- 2 Right-click the Saved Queries label on the menu bar, and then click Inspect Element.
- 3 Enter the following text in the search window:

```
PDQTool barContainer
```

For assistance, see the screen capture in [Step 6 on page 154](#).

- 4 In the main window of the Developer Tools window, expand the form and div elements that occur immediately below the element you located in [Step 3](#), if necessary. For example:



- 5 Move the div element that defines the Predefined Query to a different location:
 - a Click in the Styles pane that the Developer Tools displays to the right of the main window.
 - b Add the position, top, and left attributes to the element.style block that the Developer Tools displays in the Styles pane.

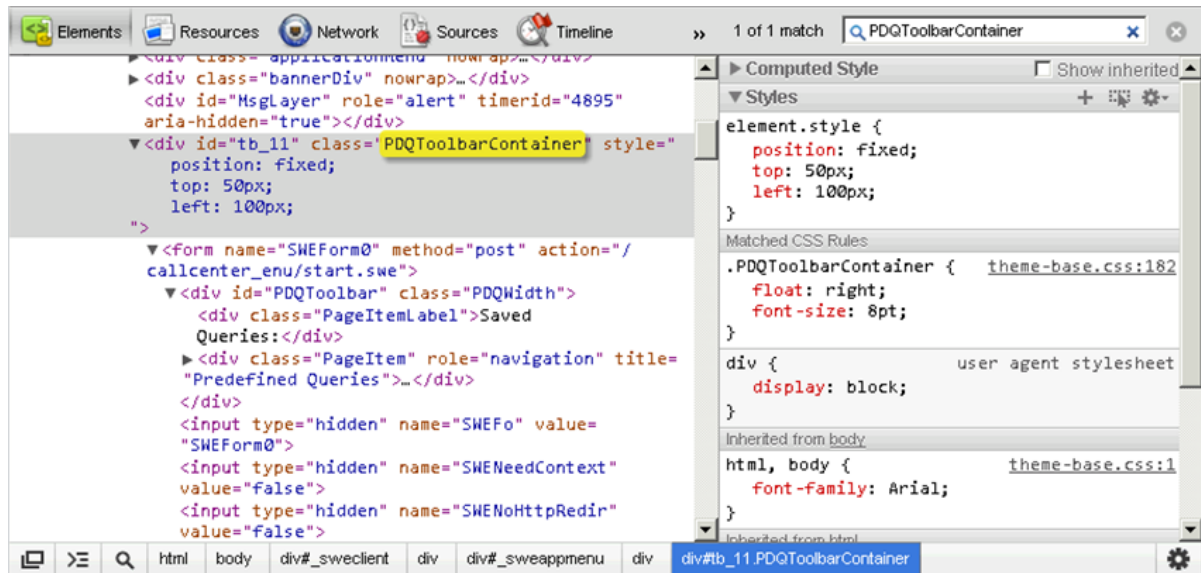
To enter these attributes, click anywhere after the left brace ({), and then start typing.

When you add these values, you modify the position, top and left properties of the PDQToolbarContainer class. Siebel Open UI applies any modification you make to a class to all elements that use the same class property.

For example, add the following attributes:


```
element.style {
  position: fixed;
  top: 38px;
  left: 270px;
}
```

Use the following screen capture for assistance:



- c Verify that Siebel Open UI positions the Predefined Query drop-down list below and to the right of the application-level Help menu.
- 6 Modify the text color of the Predefined Query label:
 - a Enter the following text in the search window:
Pagel temLabel
 - b In the main window of the Developer Tools window, click the PageItemLabel code line.
 - c Click in the Matched CSS Rules section of the Styles pane.
 - d Modify the color to red.

Using Cascading Style Sheet Classes to Modify HTML Elements

The example in this topic configures Siebel Open UI to insert several classes that the cascading style sheet uses for HTML elements that do page layout and page composition. It attaches these classes to these elements according to the page layout and page composition that Siebel Open UI modifies. This topic describes how to modify these classes. For more information, see ["Cascading Style Sheet Classes" on page 503](#).

To use cascading style sheet classes to modify HTML elements

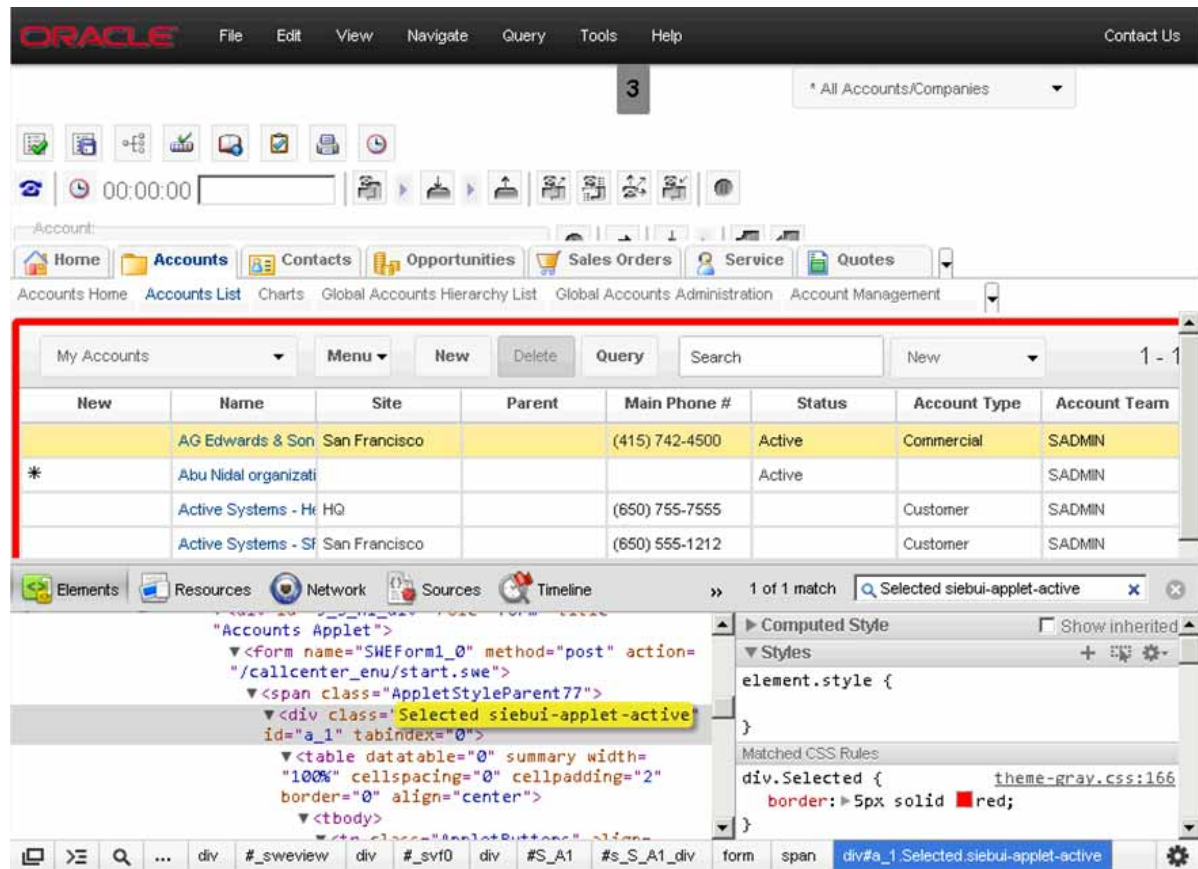
- 1 Log in to the Siebel Open UI client using Google Chrome.
- 2 Navigate to the Account screen, and then click the Account List link.
- 3 Click anywhere in the Accounts List Applet.
- 4 Right-click just to the left of the My Accounts drop-down list, and then click Inspect Element.
Siebel Open UI displays the HTML source code for the applet in a separate Developer Tools window. Notice that Siebel Open UI uses div elements in this code to define the user interface element. It does not use the frames that high interactivity uses.
- 5 Enter the following text in the search window:

Selected siebui -applet-active

The search window is located in the upper-right corner of the Developer Tools window. For assistance, see the screen capture in [Step 6](#).
- 6 In the Matched CSS Rules pane, modify the border width and the color to the following value:

border: 5px solid red

Siebel Open UI dynamically modifies the applet while you modify the source HTML. For example:



Note that this applet resides under the following s_S_A1_div div element:

```

▼<div id="s_S_A1_div" role="form" title="Accounts Applet">
  ▼<form name="SWEForm1_0" method="post" action="/callcenter_enu/start.swe">
    ▼<span class="AppletStyleParent77">
      ▼<div class="Selected siebui-applet-active" id="a_1" tabindex="0">
        ▼<table datatable="0" summary width="100%" cellpadding="2" border="0" align="center">
          ▼<tbody>
            ▼<tr class="AppletButtons" align="left">
              ►<td class="AppletTitle" nowrap>_</td>
              ►<td>_</td>
              ►<td nowrap class="AppletMenu">_</td>
            </tr>
          </tbody>
        </table>
      </div>
    </span>
  </form>
</div>

```

If you click the bottom applet, then Siebel Open UI highlights this applet and sets the class of the parent div element to s_S_A2_div. Siebel Open UI can also detach these classes in the same way.

Customizing the Sequence That Siebel Open UI Uses to Load Cascading Style Sheets

This topic describes how to replace the sequence that Siebel Open UI uses to load cascading style sheets. Siebel Open UI includes a set of cascading style sheets that it loads in a predefined sequence at run time, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4. It uses a companion Id for each style sheet immediately after it loads the main style sheet. For example, the structure cascading style sheet that Siebel Open UI loads for Siebel Mobile uses the companion sb_theme Id. Siebel Open UI loads this companion Id when it loads the theme-mb-structure.css file. The cascading style sheet that you can use to customize the structure uses the sbe_theme Id, which Siebel Open UI loads immediately after it loads sb_theme, even though it does not load a cascading style sheet file for this Id, by default. If you configure Siebel Open UI to use this Id to load a cascading style sheet file, then it loads the new cascading style sheet immediately after it loads the theme-mb-structure.css file. You can replace this sequence with your own custom sequence.

To replace the sequence that Siebel Open UI uses to load cascading style sheets

- 1 Navigate to the following folder, and then use a JavaScript editor to open the theme.js file:

`ORACLE_HOME\siebsrvr\WEBMASTER\siebel_bui\scripts\siebel\custom`

- 2 Add the following code:

```
SiebelApp.ThemeManager.addResource(  
  "GRAY_TAB",  
  {  
    css: {  
      "sbe-theme": "files/custom.css"  
    }  
  }  
);
```

where:

- `custom.css` is a .css file that includes your custom sequence. You can use any file name. For example, `my-layout.css`.

How Siebel Mobile Loads Cascading Style Sheets That It Displays on Tablets

Table 11 describes the sequence that Siebel Open UI uses to load Ids and cascading style sheet files when it renders the client in a tablet. It uses the SBL-MOBILE theme, by default. It loads sb_theme first, and then sbe_theme, and so on.

Table 11. How Siebel Open UI Loads Cascading Style Sheets That It Displays on Tablets

| Sequence | Id | File Name | Description |
|----------|-----------|---------------------------|---|
| 1 | sb_theme | theme-mb-structure.css | Predefined file that specifies the default page layout. |
| 2 | sbe_theme | Not applicable. | Placeholder that you can use to load your custom cascading style sheet to customize the default layout. |
| 3 | ss_theme | theme-mb-swatches.min.css | Predefined file that specifies color schemes. |
| 4 | sc_theme | theme-mb-style.css | File that specifies the default colors and style. |
| 5 | sce_theme | Not applicable. | Placeholder that you can use to load your custom cascading style sheet to customize the default colors and style. |

How Siebel Mobile Loads Cascading Style Sheets That It Displays on Phones

Table 12 describes the sequence that Siebel Open UI uses to load Ids and cascading style sheet files when it renders the client on a phone. It uses the SBL-MOBILE theme, by default.

Table 12. How Siebel Open UI Loads Cascading Style Sheets That It Displays on Phones

| Sequence | Id | File Name | Description |
|----------|------------|-------------------------|---|
| 1 | sb_theme | theme-mb-structure.css | Predefined file that specifies the default page layout. |
| 2 | sbe_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that modifies the default layout. |
| 3 | sbp_theme | theme-mbp-structure.css | Predefined file that specifies the page layout that Siebel Open UI displays in a phone. |
| 4 | sbpe_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that modifies the default layout that Siebel Open UI displays in a phone. |

Table 12. How Siebel Open UI Loads Cascading Style Sheets That It Displays on Phones

| Sequence | Id | File Name | Description |
|----------|------------|---------------------------|--|
| 5 | ss_theme | theme-mb-swatches.min.css | Predefined file that specifies color schemes. |
| 6 | sc_theme | theme-mb-style.css | File that specifies the default colors and style. |
| 7 | sce_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that specifies colors and style. |
| 8 | scp_theme | theme-mbp-style.css | Predefined file that specifies default colors and style for a phone. |
| 9 | scpe_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that modifies the default color and style that Siebel Open UI displays in a phone. |

How Siebel Mobile Loads Cascading Style Sheets That It Displays on Desktops

Table 13 describes the sequence that Siebel Open UI uses to load Ids and cascading style sheet files when it renders the client on a desktop computer. You can use ThemeRoller swatches only with Siebel Mobile. However, you can use the files that Table 13 describes to customize how Siebel Open UI loads the cascading style sheets that it displays on desktop computers for the GRAY_TAB theme. It uses the same loading sequence and Ids for all themes. Only the file names are specific to a theme.

Table 13. How Siebel Open UI Loads Cascading Style Sheets That It Displays on Desktops

| Sequence | Id | File Name | Description |
|----------|-----------|-------------------|--|
| 1 | sb_theme | theme-base.css | Predefined file that specifies the page layout. |
| 2 | sbe_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that modifies the default layout. |
| 3 | sc_theme | theme-gray.css | Predefined file that specifies color schemes and style. |
| 4 | sce_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that modifies the default color schemes and style. |
| 5 | sn_theme | theme-nav-tab.css | Predefined file that specifies to use tabs for navigation. |

Table 13. How Siebel Open UI Loads Cascading Style Sheets That It Displays on Desktops

| Sequence | Id | File Name | Description |
|----------|------------|--------------------|--|
| 6 | sne_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that specifies navigation. |
| 7 | sca_theme | theme-calendar.css | Predefined file that specifies styles in the calendar. |
| 8 | scae_theme | Not applicable | Placeholder that you can use to load a custom cascading style sheet that specifies styles in the calendar. |

Customizing Applets

This topic describes how to customize applets. It includes the following information:

- [Refreshing Applets That Contain Modified Data on page 159](#)
- [Allowing Users to Drag and Drop Data Into List Applets on page 163](#)
- [Customizing List Applets to Display a Box List on page 164](#)
- [Customizing List Applets to Render as a Carousel on page 167](#)
- [Customizing List Applets to Render as a Carousel without Compiling the SRF on page 173](#)
- [Customizing List Applets to Render as a Table on page 176](#)
- [Configuring the Focus in List Applets on page 179](#)
- [Adding Static Drilldowns to Applets on page 180](#)
- [Allowing Users to Change the Applet Visualization on page 182](#)
- [Displaying Applets Differently According to the Applet Mode on page 190](#)
- [Adding Custom User Preferences to Applets on page 196](#)
- [Customizing Applets to Capture Signatures on page 199](#)

Refreshing Applets That Contain Modified Data

You can configure Siebel Open UI to refresh only the applet that includes data that Siebel Open UI modified. This configuration makes rendering in the client more efficient than refreshing all applets, including applets that do not contain any new data. The example in this topic configures Siebel Open UI to allow the user to dynamically toggle the following applets in a view:

- One applet displays internal Siebel CRM data that Siebel Open UI gets from the Siebel Server.
- One applet displays external data that Siebel Open UI gets from some other, non-Siebel Web site.

To refresh applets that contain modified data**1** Copy the JavaScript files:

- a** Download a copy of the partialrefreshpm.js file to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. For more information about this file, see [“Example of a Presentation Model” on page 42](#).

- b** Download a copy of the partialrefreshpr.js file to in the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. For more information about this file, see [“Example of a Physical Renderer” on page 43](#).

2 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.

For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- b** Navigate to the Administration - Application screen, and then the Manifest Files view.

- c** In the Files list, add the following files.

| Field | Value |
|-------|-----------------------------------|
| Name | siebel/custom/partialrefreshpr.js |
| Name | siebel/custom/partialrefreshpm.js |

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.

- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Contact Form Applet |

- f** In the Object Expression list, add the following expression. The physical renderer uses this expression to render the applet in a mobile platform.

| Field | Value |
|------------|--------|
| Expression | Mobile |
| Level | 1 |

- g** In the Files list, add the following file:
siebel/custom/partialrefreshpr.js
- h** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Presentation Model |
| Name | Contact Form Applet |

- i** In the Object Expression list, add a record with no value in the Expression field.
 - j** In the Files list, add the following file:
siebel/custom/partialrefreshpm.js
- 3** Test your modifications:
- a** Open the browser in the client computer, and then clear the browser cache.
For more information, see [“Clearing the Browser Cache” on page 204](#).
 - b** Open the Siebel application, and then navigate to the Contact Form Applet.
 - c** Delete the value in the Job Title field, and then step out of the field.
 - d** Make sure Siebel Open UI removes the values from the Work # and the Main Fax # fields.
 - e** Add a value to the Job Title field, and then step out of the field.
 - f** Make sure Siebel Open UI adds values to the Work # and the Main Fax # fields.

Text Copy of Code That Does a Partial Refresh for the Presentation Model

To get a copy of the partialrefreshpm.js file, see Article ID 1494998.1 on My Oracle Support. If you do not have access to this file on My Oracle Support, then you can open a JavaScript editor, create a new file named partialrefreshpm.js, copy the following code into this file, and then save your modifications:

```
if(typeof(SiebelAppFacade.PartialRefreshPM) === "undefined"){

    SiebelJS.Namespace("SiebelAppFacade.PartialRefreshPM");
    define("siebel/custom/partialrefreshpm", [], function () {(
    SiebelAppFacade.PartialRefreshPM = (function(){
        function PartialRefreshPM(proxy){
            SiebelAppFacade.PartialRefreshPM.superclass.constructor.call(this, proxy);
        }
        SiebelJS.Extend(PartialRefreshPM, SiebelAppFacade.PresentationModel);
        PartialRefreshPM.prototype.Init = function(){
            SiebelAppFacade.PartialRefreshPM.superclass.Init.call(this);
            this.AddProperty("ShowJobTitleRelatedField", "");
            this.AddMethod("ShowSelection", SelectionChange, {sequence : false, scope :
```

```

this.s});
    this.s.AddMethod("FieldChange", OnFieldChange, {sequence : false, scope: this.s});
};
function SelectonChange(){
    var controls = this.s.Get("GetControls");
    var control = controls[ "JobTitle" ];
    var value = this.s.ExecuteMethod("GetFieldValue", control);
    this.s.SetProperty("ShowJobTitleRelatedField", (value ? true: false));
}
function OnFieldChange(control, value){
    if(control.GetName() === "JobTitle"){
        this.s.SetProperty("ShowJobTitleRelatedField", (value ? true: false));
    }
}
return PartialRefreshPM;
}());
}

```

Text Copy of Code That Does a Partial Refresh for the Physical Renderer

To get a copy of the partialrefreshpr.js file, see Article ID 1494998.1 on My Oracle Support. If you do not have access to this file on My Oracle Support, then you can open a JavaScript editor, create a new file named partialrefreshpr.js, copy the following code into this file, and then save your modifications:

```

if(typeof(SiebelAppFacade.PartialRefreshPR) === "undefined"){
    SiebelJS.Namespace("SiebelAppFacade.PartialRefreshPR");
    define("siebel/custom/partialrefreshpr", ["order!3rdParty/jquery.siegnaturepad.min",
    "order!siebel/phyrenderer"], function () {
    SiebelAppFacade.PartialRefreshPR = (function(){
        function PartialRefreshPR(pm){
            SiebelAppFacade.PartialRefreshPR.superclass.constructor.call(this, pm);
        }
        SiebelJS.Extend(PartialRefreshPR, SiebelAppFacade.PhysicalRenderer);
        PartialRefreshPR.prototype.Init = function () {
            SiebelAppFacade.PartialRefreshPR.superclass.Init.call(this);
            this.AttachPMBinding("ShowJobTitleRelatedField", ModifyLayout);
        };
        function ModifyLayout(){
            var controls = this.s.GetPM().Get("GetControls");
            var canShow = this.s.GetPM().Get("ShowJobTitleRelatedField");
            var WorkPhoneNum = controls[ "WorkPhoneNum" ];
            var FaxPhoneNum = controls[ "FaxPhoneNum" ];
            if(canShow){
                $("#div#WorkPhoneNum_Label").show();
                $("[name=' " + WorkPhoneNum.GetInputName() + "']").show();
                $("#div#FaxPhoneNum_Label").show();
                $("[name=' " + FaxPhoneNum.GetInputName() + "']").show();
            }
            else{
                $("#div#WorkPhoneNum_Label").hide();
                $("[name=' " + WorkPhoneNum.GetInputName() + "']").hide();
            }
        }
    })();
}

```

```

        $("div#FaxPhoneNum_Label").hide();
        $("[name=' " + FaxPhoneNum.GetInputName() + "']").hide();
    }
    }
    return PartialRefreshPR;
} ());
return "Siebel AppFacade. Partial RefreshPR";
});
}

```

Allowing Users to Drag and Drop Data Into List Applets

The example in this topic describes how to allow users to drag and drop data from a spreadsheet to the Contact List applet.

To allow users to drag and drop data into list applets

1 Modify the applet:

a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

b In the Object Explorer, click Applet.

c In the Applets list, query the Name property for Contact List Applet.

d In the Object Explorer, expand the Applet tree, and then click Applet User Prop.

e In the Applet User Properties list, add the following applet user properties.

| Name | Value |
|-------------------------|-------------------------|
| ClientPMUserProp | EnableDragAndDropInList |
| EnableDragAndDropInList | TRUE |

f Compile your modifications.

2 Identify the columns that you must drag and drop:

a Log in to the client, navigate to the Contacts screen, and then the Contacts List.

b In the contact form, notice the required fields.

Siebel Open UI uses a red asterisk to indicate each required field. In the contact form, the Last Name and First Name fields are required.

3 Create a spreadsheet:

a Open a spreadsheet application, such as Microsoft Excel.

b In the first row, add the column headers for the columns that you must drag and drop.

- ❑ For each column name that you include, make sure the column name is identical to the column name that the list applet displays in the client.
- ❑ Siebel Open UI does not require you to include all column headers. However, you must include all the required column headers that you noticed in [Step 2](#).
- ❑ You can include column headers in any order.
- c Add data rows immediately below the column header row that you added in [Step b](#).
For example, add rows that include information about each contact, such as first name and last name.

Your completed work might resemble the following spreadsheet:

| | A | B | C | D |
|---|------------|-----------|------------------|-------|
| 1 | First Name | Last Name | Account | Mr/Ms |
| 2 | Antonia | Pinas | Partner PC Local | Ms. |
| 3 | Mary | Aaron | Atherton Group | Mrs. |
| 4 | Diana | Abbot | Abbot Designs | Ms. |

- 4 Drag and drop the data:
 - a In the spreadsheet application, choose the cells that include the header and data information.
 - b Drag and drop the cells that you chose in [Step a](#) to the Contact List Applet in the Siebel application.
Do the following to drag and drop cells in Excel. Your spreadsheet program might work differently:
 - ❑ Position the cursor over a corner of the selection area until Excel displays the cursor as a four-way arrow.
 - ❑ Right-click and hold down the right mouse button over the cursor.
 - ❑ Drag the selection area to the Contact List Applet.
 - ❑ Release the mouse button.
 - c Verify that Siebel Open UI added the data rows to the list applet.

Customizing List Applets to Display a Box List

This topic describes how to customize a list applet to display a box list. You customize how Siebel Open UI renders an applet, the content it displays, and the style that it uses in the client.

To customize list applets to display a box list

- 1 Log in to the client.

- 2 Navigate to a view that displays a typical Siebel list applet.

For example, navigate to the Accounts screen, and then the Accounts list.

Notice that Siebel Open UI displays the typical predefined list.

- 3 Open Windows Explorer, and then navigate to the following folder:

```
\release_number\eappweb\PUBLIC\language_code\release_number\scripts\siebel
```

For example:

```
C:\23013\eappweb\PUBLIC\enu\23013\scripts\siebel
```

- 4 Rename the existing jqgridrenderer.js file that resides in the folder you accessed in [Step 3](#) to jqgridrenderer_original.js.

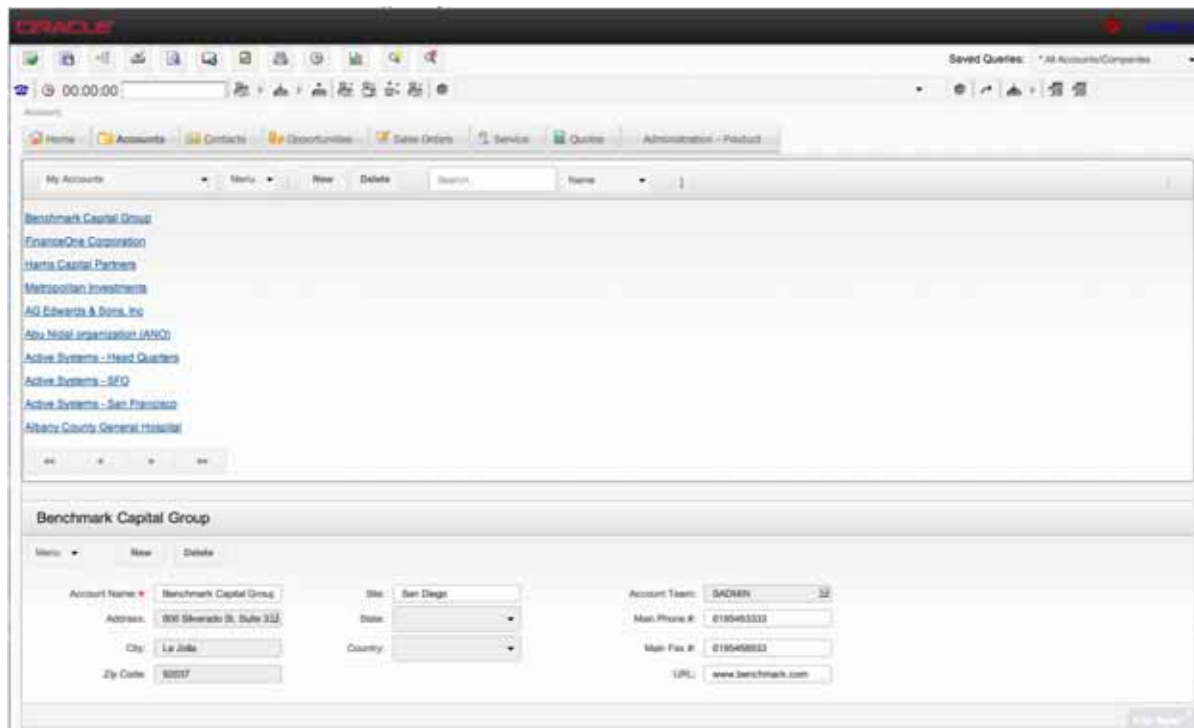
- 5 Download the jqgridrenderer_tile.js file to the folder you accessed in [Step 3](#).

The jqgridrenderer_tile file prevents Siebel Open UI from initializing the jqgrid control and from rendering other fields in the grid. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

- 6 Rename the jqgridrenderer_tile.js file to jqgridrenderer.js.

- 7 In the Siebel Open UI client, press the F5 key to refresh the screen.

The client displays the modified layout. For example:



- 8 In Windows Explorer, navigate to the following folder:

CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom

9 Use an editor to open the theme_base.css file.

10 Copy the following code into the theme_base.css file. This code configures Siebel Open UI to display account names in a series of vertical boxes:

```
/*-----*/
/* Styles for alternate List display demo */
/*-----*/
.siebui-boxlist {
    width: 100%;
    height: 100%;
    overflow: auto;
}
.siebui-boxlist-pager, .siebui-boxlist-items{
    display: table-row;
    white-space: nowrap;
    width: 100%;
}
.siebui-boxlist-item, siebui-boxlist-item-selected {
    padding: 100px 0px;
    height: 40px;
    border-radius: 5px;
    float: left;
    width: 120px;
    overflow: hidden;
    margin: 5px 12px;
    color: #222!important;
    text-shadow: 0 1px 0 rgba(255, 255, 255, 0.7);
    text-align: center;
}
.siebui-boxlist-item {
    background: rgb(250, 250, 250);
    background: -moz-linear-gradient(top, rgba(250, 250, 250, 1) 0%, rgba(225, 225, 225, 1) 100%);
    background: -webkit-gradient(linear, left top, left bottom, color-stop(0%, rgba(250, 250, 250, 1)), color-stop(100%, rgba(225, 225, 225, 1)));
    background: -webkit-linear-gradient(top, rgba(250, 250, 250, 1) 0%, rgba(225, 225, 225, 1) 100%);
    border-bottom: 1px solid #AAA;
    box-shadow: 0 0 3px rgba(0, 0, 0, 0.4);
    -webkit-box-shadow: 0 0 3px rgba(0, 0, 0, 0.4);
}
.siebui-boxlist-item-selected {
    background: rgb(250, 250, 250);
    background: -moz-linear-gradient(top, rgba(249, 238, 167, 0.5) 0%, rgba(251, 236, 136, 0.5) 100%)!important;
    background: -webkit-gradient(linear, left top, left bottom, color-stop(0%, rgba(249, 238, 167, 0.5)), color-stop(100%, rgba(251, 236, 136, 0.5)))!important;
    background: -webkit-linear-gradient(top, rgba(249, 238, 167, 0.5) 0%, rgba(251, 236, 136, 0.5) 100%)!important;
    border-bottom: 1px solid #AAA;
    box-shadow: 0 0 3px rgba(0, 0, 0, 0.4);
    -webkit-box-shadow: 0 0 3px rgba(0, 0, 0, 0.4);
}
```

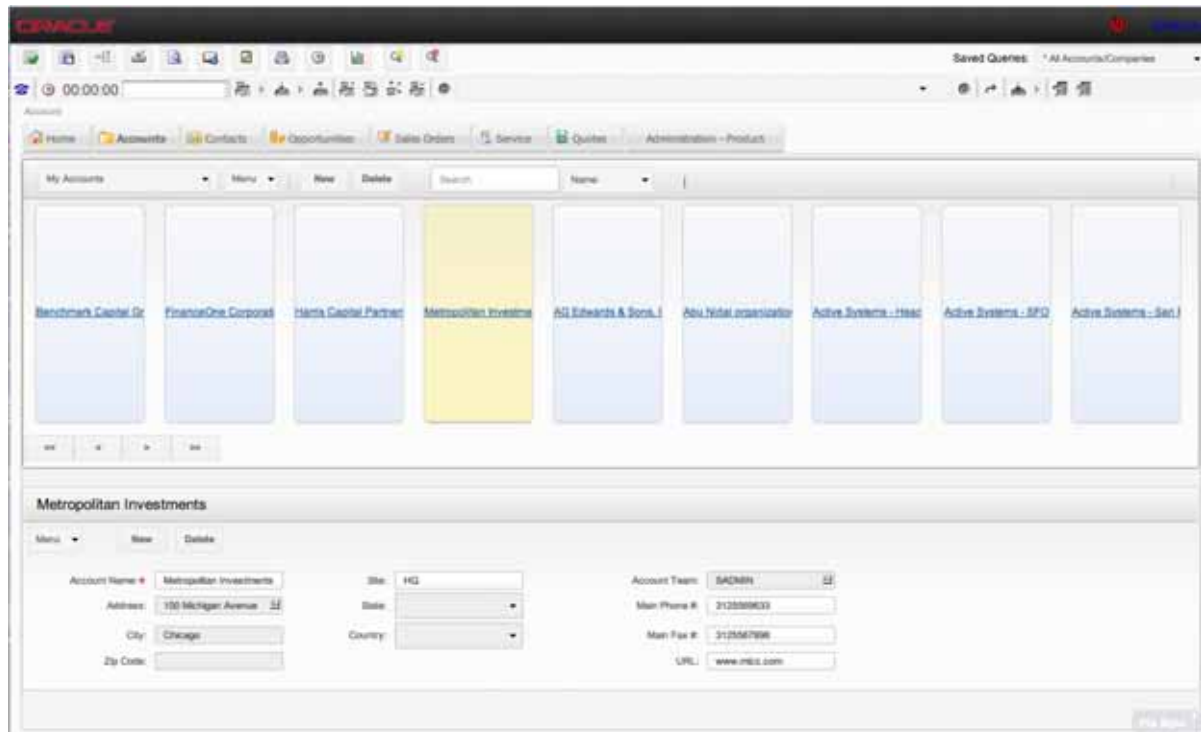
```

}
/*-----*/
/* Styles for alternate List display demo */
/*-----*/

```

- 11 Navigate to the Siebel Open UI client, and then press the F5 key to refresh the screen.

The client displays the modified layout. For example:



Customizing List Applets to Render as a Carousel

The example in this topic describes how to customize Siebel Open UI to render a list applet as a carousel. To view different example carousel styles and get source code for these styles, see the <http://sorgalla.com/projects/jcarousel> Web site. For an alternative configuration, see “[Customizing List Applets to Render as a Carousel without Compiling the SRF](#)” on page 173.

To customize list applets to render as a carousel

- 1 Add records in the client:
 - a Make sure the application configuration file for Siebel Call Center includes the following setting:


```
[Siebel]
RepositoryFile = siebel_sia.srf
```

For more information, see “[Modifying the Application Configuration File](#)” on page 105.

- b** Open the client, navigate to the Contacts screen, and then click the Contact List link.
- c** Add the following contact.

| Field | Value |
|------------|-------|
| Last Name | Aamos |
| First Name | Ray |

- d** Click the link in the Last Name.
 - e** Click the Affiliations link.
 - f** In the Affiliations list, add four affiliations.
 - g** Make sure you choose a different value in the Account field for each record. Accept default values for all other fields.
 - h** Log out of the client.
- 2** Modify the JavaScript files:

- a** Copy the carouselrenderer.js file to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

The carouselrenderer.js file is a physical renderer that bridges a JCarousel third-party control plug-in to the list presentation model that the listpmodel.js file defines. The List Applet and the Carousel applet use the same presentation model as the business logic for each user interface representation. The only difference is how Siebel Open UI renders each applet.

- b** Download the jquery.jcarousel.js file into the following folder:

`CLIENT_HOME\PUBLIC\language_code\release_number\scripts\3rdParty`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. Oracle downloads and integrates this 3rdParty Carousel package into Siebel Open UI through the physical renderer. You must never modify these third-party plug-in files. If you require a configuration that the third-party plug-in does not meet, then you must modify the physical renderer instead of the third-party plug-in.

- 3** Configure Siebel Open UI to use the CSS file that the third-party uses:

- a** In Windows Explorer, navigate to the following folder:

`CLIENT_HOME\PUBLIC\language_code\release_number\scripts\3rdParty`

- b** Create the jcarousel, skins, and tango subfolders in the 3rdParty folder using the following hierarchy:

`CLIENT_HOME\PUBLIC\language_code\release_number\scripts\3rdParty\jcarousel\skins\tango\`

- c** Download the skin.css file into the following folder:

CLIENT_HOME\PUBLIC\language_code\release_number\scripts\3rdParty\jcarousel\skins\tango

To get a copy of the skin.css file, see Article ID 1494998.1 on My Oracle Support.

- d** Navigate to the following folder:

CLIENT_HOME\PUBLIC\language_code\release_number\scripts\siebel

- e** Use a JavaScript editor to open the theme.js file.

- f** Modify the following code. You add the jcarousel code lines. Make sure you add a comma to the end of the code line that immediately precedes the jcarousel line. If you copy and paste code, make sure your JavaScript editor pastes straight quotation marks ("), not smart quotation marks ("):

```
/*global $ SIEBEL_BUID */
SiebelApp.ThemeManager.addTheme(
  "GRAY_TAB",
  {
    css : {
      "sb_theme" : "files/theme-base.css",
      "sc_theme" : "files/theme-gray.css",
      "sn_theme" : "files/theme-nav-tab.css",
      "sca_theme" : "files/theme-calendar.css",
      "jcarousel" : "SIEBEL_BUID + "3rdParty/jcarousel/skins/tango/skin.css"
    },
    objList : [ ]
  }
);

SiebelApp.ThemeManager.addTheme(
  "TANGERINE_TAB",
  {
    css : {
      "sb_theme" : "files/theme-base.css",
      "sc_theme" : "files/theme-tangerine.css",
      "sn_theme" : "files/theme-nav-tab.css",
      "sca_theme" : "files/theme-calendar.css",
      "jcarousel" : "SIEBEL_BUID + "3rdParty/jcarousel/skins/tango/skin.css"
    },
    objList : [ ]
  }
);
```

- g** Use a JavaScript editor to open the themetree.js file.

- h** Modify the following code. You add the jcarousel code lines:

```
/*global $ SIEBEL_BUID */
SiebelApp.ThemeManager.addTheme(
  "GRAY_ACCORDION",
  {
    css : {
      "sb_theme" : "files/theme-base.css",
      "sc_theme" : "files/theme-gray.css",
```

```

        "sn_theme" : "files/theme-nav-accordion.css",
        "sca_theme" : "files/theme-calendar.css",
        "dyt_theme" : "files/3rdParty/themes/dynatree/skin/ui.dynatree.css"
        "j_carousel" : "SIEBEL_BUI LD + "3rdParty/j_carousel /skins/tango/skin.css"
    },
    objList : [ ]
}
);

SiebelApp.ThemeManager.addTheme(
    "TANGIERNE_ACCORDION",
    {
        css : {
            "sb_theme" : "files/theme-base.css",
            "sc_theme" : "files/theme-tangerine.css",
            "sn_theme" : "files/theme-nav-accordion.css",
            "sca_theme" : "files/theme-calendar.css",
            "dyt_theme" : "files/3rdParty/themes/dynatree/skin/ui.dynatree.css"
            "j_carousel" : "SIEBEL_BUI LD + "3rdParty/j_carousel /skins/tango/skin.css"
        },
        objList : [ ]
    }
);

```

4 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.

For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- b** Navigate to the Administration - Application screen, and then the Manifest Files view.
- c** In the Files list, add the following file.

| Field | Value |
|-------|-----------------------------------|
| Name | siebel/custom/carouselrenderer.js |

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.
- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|--|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Pharma Professional Affiliation From List Applet |

- f** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 1 |

- g** In the Files list, add the following file:
- siebel /custom/carousel renderer. j s
- 5** Test your modifications:
- a** Clear the browser cache.
- For more information, see ["Clearing the Browser Cache" on page 204](#).
- b** Open the Siebel application, and then navigate to the contact that includes the affiliations that you added in [Step 1](#).

- c Make sure the affiliations view is similar to the following.

Notice that the carousel data runs together because no styling is defined for the carousel content. To fix this problem, do [Step 6](#).

- 6 Modify the styling that Siebel Open UI uses to render the view:

- a Use a JavaScript editor to open the carouselrenderer.js file that you copied in [Step 2](#).

- b Locate the following code:

```
i temMarkup += "</span><br>";
```

- c Modify the code you located in [Step b](#) to the following. You remove the break:

```
i temMarkup += "</span>";
```

- d Use a JavaScript editor to open the skin.css file.

- e Locate the following code:

```
.j carousel -skin-tango .j carousel -i tem {
  width: 75px;
  height: 75px;
}
```

- f** Modify the code you located in [Step e](#) to the following. Bold font indicates the code that you must modify:

```
.j carousel -skin-tango .j carousel -item {
  width: 318px;
  height: 75px;
}
```

- g** Locate the following code:

```
.j carousel -skin-tango .j carousel -item-horizontal {
  margin-left: 0;
  margin-right: 10px;
}
```

- h** Modify the code you located in [Step g](#) to the following. Bold font indicates the code that you must modify:

```
.j carousel -skin-tango .j carousel -item-horizontal {
  margin-left: 10;
  margin-right: 10px;
  color: black;
}
```

- 7** Test your modifications:

- a** Clear the browser cache.
- b** Refresh the view that you examined in [Step 5](#).
- c** Make sure the styling is similar to the following.



Customizing List Applets to Render as a Carousel without Compiling the SRF

This topic describes how to modify the appearance and behavior of the user interface without compiling the SRF. As an example, it describes how to configure Siebel Open UI to render a list applet as a carousel without compiling the SRF. It gets information from LinkedIn, a 3rd party resource.

To customize list applets to render as a carousel without compiling the SRF

- 1** Download the sociallyawarerenderer.js file into the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

- 2 Locate the following code in the `sociallyawarerenderer.js` file:

`Siebel JS. Extend(SociallyAwarePR, Siebel AppFacade. Physical Renderer);`

- 3 Add the following code immediately below the code you located in [Step 2](#). The code that you add in this step adds the rendering implementation methods:

```
functionGetCurrentCarouselItems(dbRec, linkedInRec){
    varpm=this.GetPM();
    varlistCols=pm.Get("ListOfColumns");
    varitemMarkup="<div>";
    itemMarkup+="

```

- 4 Locate the following code in the `sociallyawarerenderer.js` file:

`Siebel JS. Extend(SociallyAwarePR, Siebel AppFacade. Physical Renderer);`

- 5 Add the following code immediately below the code you located in [Step 4](#):

```
SociallyAwarePR.prototype.ShowUI = function(){
    Siebel AppFacade. SociallyAwarePR. superclass.ShowUI.call(this);
    var pm = this.GetPM();
    var placeholder = pm.Get("GetPlaceholder");
    $("#"+placeholder).append("<ul class='siebui-list-carousel jcarousel-skin-
tango'></ul>");
    var recordSet = pm.Get("GetRecordSet");
    var recordLength = recordSet.length;
```

```

var linkedInRecSet = pm.Get("linkedInRecordSet");
var markup = "";
for(var i = 0; i < recordLength; i++){
    markup += "<li>" + GetCurrentCarouselItems.call(
        this,
        recordSet[i],
        (pm.ExecuteMethod("getConnectionByName", recordSet[i]["First Name"],
        recordSet[i]["Last Name"]) || {})) + "</li>";
}
$("ul.siebui-list-carousel", "#" +
placeholder).html("").append(markup).jcarousel({
    itemFallbackDimension: 300
});
};

```

- 6 Locate the following code in the `sociallyawarerenderer.js` file:

```

function GetControlIndex(key, listCols){
    var result = -1;
    for(var i = 0; i < listCols.length; i++){
        if(listCols[i].name === key){
            result = i;
            break;
        }
    }
    return result;
}

```

- 7 Add the following code immediately below the code you located in [Step 6](#):

```

SociallyAwarePR.prototype.BindData = function(){
    var pm = this.GetPM();
    var placeholder = pm.Get("GetPlaceholder");
    var recordSet = pm.Get("GetRecordSet");
    var recordLength = recordSet.length;
    var linkedInRecSet = pm.Get("linkedInRecordSet");
    var markup = "";
    for(var i = 0; i < recordLength; i++){
        markup += "<li>" + GetCurrentCarouselItems.call(
            this,
            recordSet[i],
            (pm.ExecuteMethod("getConnectionByName", recordSet[i]["First Name"],
            recordSet[i]["Last Name"]) || {})) + "</li>";
    }
    $("ul.siebui-list-carousel", "#" +
    placeholder).html("").append(markup).jcarousel({
        itemFallbackDimension: 300
    });
};

```

- 8 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.

For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- b** Navigate to the Administration - Application screen, and then the Manifest Files view.
- c** In the Files list, add the following file.

| Field | Value |
|-------|--|
| Name | siebel/custom/sociallyawarerenderer.js |

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.
- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|--|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Pharma Professional Affiliation From List Applet |

- f** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 1 |

- g** In the Files list, add the following file:

si ebel /custom/soci al l yawarerenderer. j s

- 9** Test your modifications.

Customizing List Applets to Render as a Table

The example in this topic describes how to customize a list applet to display as a table. The *JQM Grid Renderer* is an object that uses the `jqmgridrenderer.js` file to render data in a table in the client. It does the following:

- Renders controls according to the control type that the Siebel Server sends to the client. For example, a table can include a combo box, pick list, check box, date and time control, text box, and so on.
- Allows the user to use a table to edit data.

- Comes predefined to do an implicit save that saves data if the user steps out of a row to another row in the table.
- Allows the user to click a column heading to sort data in the table in ascending or descending order.
- Includes a Delete button that allows the user to delete each row of the table.
- Highlights the currently chosen row in a different color.
- Includes 4 or 5 columns.
- Displays some fields as text, such as the URL, mail to address, or phone number.

To customize list applets to render as a table

- 1 Identify the list applet that you must modify. For example:
 - a Log in to the client, and then navigate to the Products screen.
 - b Click Help, and then click About View.
 - c Make a note of the applet name.
For example, SIS Product List Applet.
- 2 Configure the manifest:
 - a Log in to the Siebel client with administrative privileges.
For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).
 - b Navigate to the Administration - Application screen, and then the Manifest Files view.
 - c In the Files list, add the following files.

| Field | Value |
|-------|---------------------------------|
| Name | siebel/jqmgridrenderer.js |
| Name | siebel/jqmtabletgridrenderer.js |
| Name | siebel/jqgridrenderer.js |

- d Navigate to the Administration - Application screen, and then the Manifest Administration view.
- e In the UI Objects list, specify the following applet.

| Field | Value |
|------------|-------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | My Applet |

- f** In the Object Expression list, add the following expression. The physical renderer uses this expression to render the applet on a phone.

| Field | Value |
|------------|-------|
| Expression | Phone |
| Level | 1 |

- g** In the Files list, add the following file:

`si_ebel /j_qmgri_drenderer.js`

- h** In the Object Expression list, add the following expression. The physical renderer uses this expression to render the applet on a tablet.

| Field | Value |
|------------|--------|
| Expression | Tablet |
| Level | 2 |

- i** In the Files list, add the following file:

`si_ebel /j_qmtabletgr_i_drenderer.js`

- j** In the Object Expression list, add the following expression. The physical renderer uses this expression to render the applet on a desktop.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 3 |

- k** In the Files list, add the following file:

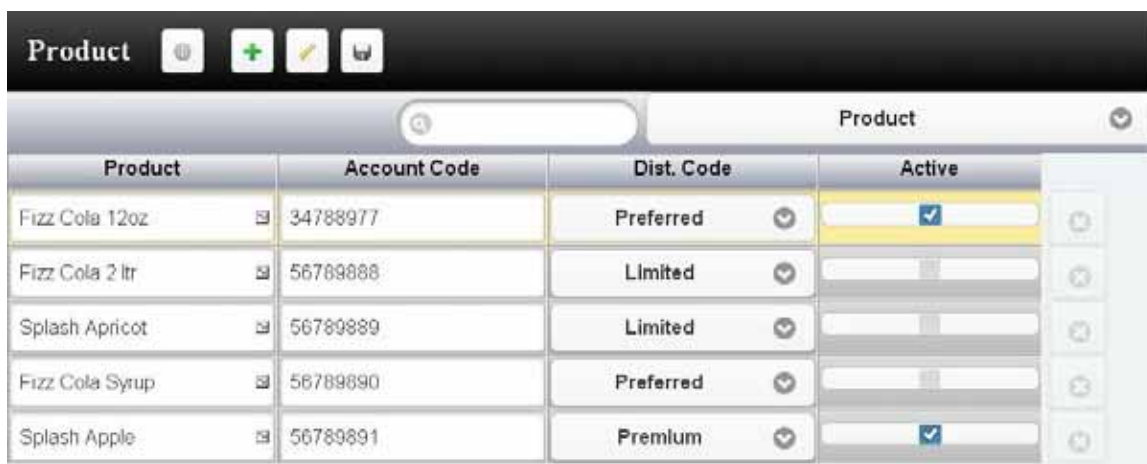
`si_ebel /j_qgri_drenderer.js`

- 3** Verify your work:

- a** Log in to the client, and then navigate to the Products screen.

- b** Make sure Siebel Open UI displays the SIS Product List Applet as a table.

For example, in a mobile application, make sure the result is similar to the following screen capture:



| Product | Account Code | Dist. Code | Active |
|-----------------|--------------|------------|-------------------------------------|
| Fizz Cola 12oz | 34788977 | Preferred | <input checked="" type="checkbox"/> |
| Fizz Cola 2 ltr | 56789888 | Limited | <input type="checkbox"/> |
| Splash Apricot | 56789889 | Limited | <input type="checkbox"/> |
| Fizz Cola Syrup | 56789890 | Preferred | <input type="checkbox"/> |
| Splash Apple | 56789891 | Premium | <input checked="" type="checkbox"/> |

Configuring the Focus in List Applets

If you modify a list applet, then you must make sure that your modification does not adversely affect how Siebel Open UI sets the focus in this list applet. Siebel Open UI does the following work to set the focus in a list applet:

- 1 Sets the focus to the list column that includes a list column user property that specifies a default focus, such as `DefaultFocus_Edit`. This list column is a child object type of the list applet. For more information about default focus user properties, see the topic about Specifying the Default Applet Focus in *Siebel Developer's Reference*.
- 2 If the list column user property described in [Step 1](#) does not exist, then Siebel Open UI examines the columns of a row from left to right, and then places the focus on the first editable control that it encounters. It continues examining rows in this way until it finds an editable control, or until it reaches the last column of the last row.
- 3 If Siebel Open UI does not find any editable controls in [Step 2](#), then it sets the focus on the first noneditable control that the list applet displays.
- 4 If Siebel Open UI does not find any noneditable controls in [Step 3](#), then it sets the focus on the div container that it uses to display the list applet.

Assume you do the following configuration:

- Use Siebel Tools to add a large number of list columns to the SIS Account List Applet.
- Make all list columns except the last list column read-only, and then compile the SRF.

- Log in to the client, navigate to the Account list view, and then run a query.

In this situation, Siebel Open UI places the focus on the last list column that the list applet contains. The div container might not contain enough room to display this list column, the list column might not be visible in the applet, and you might not be able to use the applet because the focus is on a column that you cannot access.

To configure the focus in list applets

- Make sure your configuration does not set the focus to a list column or field that Siebel Open UI displays only partially or does not display at all.

You can use the following guidelines:

- If Siebel Open UI sets the focus to a list column that contains a DefaultFocus list column user property, then make sure it correctly displays this list column after you finish your modifications.
- If Siebel Open UI sets the focus to an editable or noneditable control, then make sure Siebel Open UI correctly displays this control after you finish your modifications.

To follow these guidelines, it might be necessary for you to rearrange the top-to-bottom sequence that Siebel Open UI uses to display list columns and controls in the list applet.

Adding Static Drilldowns to Applets

This topic describes how to add a static drilldown to a form applet so that the drilldown object displays the name of the destination field, such as the primary account name, in the popup label when the user clicks a drilldown link. If you do not do this configuration on a custom form applet that you create, then the drilldown link displays the data from the field as the label, such as the account name, and not the caption text from the control. For information about how to configure a drilldown on a form applet, see Article ID 539183.1 on My Oracle Support.

To add static drilldowns to applets

- 1 Create a static drilldown object on the applet that you must modify:
 - a Open Siebel Tools.
For more information about using Siebel Tools, see *Using Siebel Tools*.
 - b In the Object Explorer, click Applet.
 - c In the Applets list, query the Name property for the applet that you must modify.
 - d In the Object Explorer, expand the Applet tree, and then click Control.

- e In the Controls list, create the following control.

| Property | Value |
|-----------|--|
| Field | Specify the same field that you specified in the Hyperlink Field property of the drilldown object that you created in Step a . |
| HTML Type | Text |

For more information, see the topics about creating static drilldown objects in *Configuring Siebel Business Applications*.

- f In the Object Explorer, click Applet.
- g In the Applets list, right-click the record of the applet you are modifying, and then choose Edit Web Layout.
- h Add the control that you created in [Step e](#) to the layout.
- i Compile your modifications.
- 2 Test your modifications:
- a Log in to the client.
- b Navigate to the applet you modified, and then make sure it displays your new static drilldown object with the correct label.

For example, the following screen capture includes the correct Last Name label, and it displays the correct data in the field. If you do not do the configuration that this topic describes, then Siebel Open UI might display Last Name - Drilldown as the label and as the data in the field.

The screenshot shows the Siebel Open UI interface. At the top, there is a navigation bar with icons and labels for Home, Accounts, Contacts, and Opportunities. Below this, there is a sub-navigation bar with labels for Contacts Home, Contacts List, Consumers List, and Personal Contacts. The main content area displays a contact record for Antonia Pinas. The contact's name is shown at the top. Below the name, there is a form with several fields: a Menu dropdown, New, Delete, and Query buttons; a Last Name field with the value 'Pinas'; a First Name field with the value 'Antonia'; a Job Title field; and a Mr/Ms dropdown menu with the value 'Ms.'.

Allowing Users to Change the Applet Visualization

This topic describes how to modify an applet so that the user can change the applet visualization. The *applet visualization* is a type of configuration that specifies the layout that Siebel Open UI uses to display the applet. List, form, tile, map, grid, and carousel are each an example of an applet visualization.

Siebel Open UI allows the user to set some user preferences that determine how it displays an applet. The user can navigate to the User Preferences screen, and then use the Behavior view to set these preferences. For example, if the user chooses a value in the Visualization field of the Behavior view, such as Tile, and then navigates to a list applet that includes a tile configuration, such as the Opportunity List Applet, then Siebel Open UI displays this applet as a set of tiles. If the user clicks Grid in this applet, then Siebel Open UI displays the applet as a grid and sets Grid as the default layout only for the Opportunity List Applet. This local setting takes precedence over the global setting that the user sets in the Visualization field in the Behavior view. Siebel Open UI continues to use a tile layout for all other applets that include a tile configuration. In this situation, it displays the Opportunity List Applet as a grid even if the user logs out and then logs back in to the client.

Figure 31 includes the Contacts List that you modify in this topic so that it allows the user to change the applet visualization. It illustrates how Siebel Open UI displays this list after you successfully finish the configuration. The user can click one of the applet visualization buttons, such as Tile, to change the applet visualization.

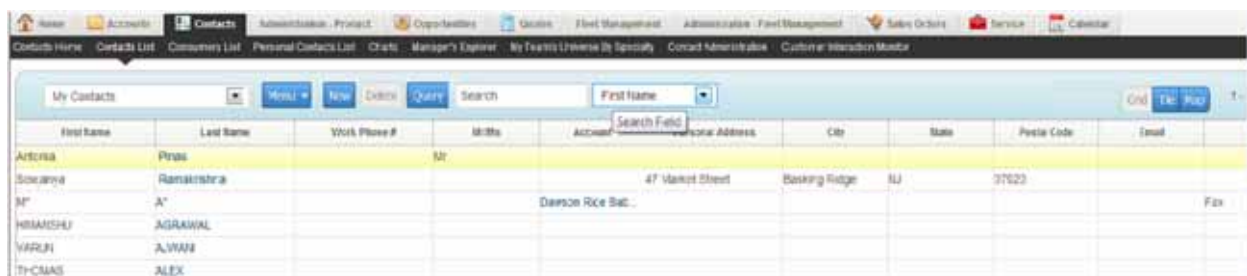


Figure 31. Contacts List That Allows Users to Change the Applet Visualization

This topic describes how to configure the manifest for a custom applet visualization. For information about configuring the manifest for a predefined configuration, see [“Configuring Manifests for Predefined Visualizations” on page 189](#).

To allow users to change the applet visualization

- 1 Modify the applet in Siebel Tools:
 - a Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b In the Object Explorer, click Applet.
 - c In the Applets list, query the Name property for the applet that you must modify.
For example, query the Name property for Contact List Applet.

- d** In the Object Explorer, expand the Applet tree, and then click Applet Web Template.

The Applet Web Templates list displays the applet modes that are defined for the applet. For example, Base, Edit, and Edit List. For more information about these modes, see [“Displaying Applets Differently According to the Applet Mode” on page 190](#).

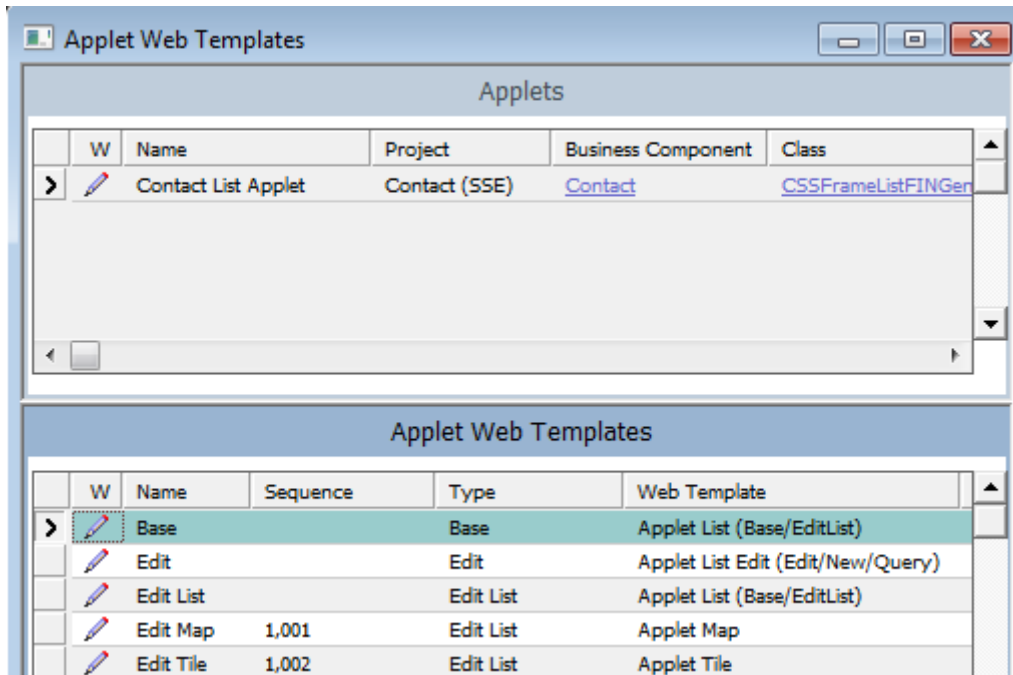
- e** In the Applet Web Templates list, add the following applet web template.

| Property | Description |
|--------------|--|
| Name | Enter text that describes the visualization behavior. For example, enter Edit Tile to describe a tile visualization that allows the user to modify field values. |
| Sequence | Enter a value of 1000 or greater. To help you quickly recognize how Siebel Open UI uses a web template, it is recommended that you use a value of: <ul style="list-style-type: none"> ■ 1000 or greater for a web template that Siebel Open UI uses to determine the applet visualization, such as a Tile. ■ 1, 2, or 3 for a web template that Siebel Open UI uses to determine the applet mode, such as Edit List. |
| Type | Specify the applet mode, such as Edit or Edit List. |
| Web Template | Choose a web template that defines the desired visualization. For example, choose Applet Tile. |

- f** Make sure Siebel Tools defines a SWT file for the web template that you defined in [Step e](#).

For example, make sure the Web Template Files list in Siebel Tools includes a record for the Applet Tile web template file, and that the FileName property for this record is CCAppletList_Tile.swt. If your deployment requires a new web template, then you must define it before you can define the applet web template. For more information about configuring web templates, see *Configuring Siebel Business Applications*.

- g** Repeat [Step d](#) and [Step e](#) for each web template that your deployment requires. Your completed work in Siebel Tools must resemble the following configuration.



- h** Compile your modifications.
- 2** Configure the manifest for the applet that you modified in [Step 1](#):
- a** Log in to the Siebel client with administrative privileges. For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).
- b** Navigate to the Administration - Application screen, and then the Manifest Files view.
- c** In the Files list, add the following predefined files.

| Field | Value |
|-------|------------------------------|
| Name | siebel/mappmodel.js |
| | siebel/Tilesrollcontainer.js |

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.

- e In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Web Template |
| Name | Contact List Applet |

- f In the Object Expression list, add the expressions that Siebel Open UI uses to render the applet for this web template in the various visualizations and applet modes that you defined in [Step 1](#).

Your completed work must resemble the following configuration. Use the Move Up, Move Down, Indent, and Outdent buttons to create the hierarchy. Note that you do not add files in the Files list for a web template. You only add files for a presentation model or physical renderer. For more information about how to create these object expressions, see ["Configuring Manifests" on page 128](#).

| Object Expression | | | | | | | 1 - 9 of 9 | |
|-------------------|------------|------------|-------|----------|-------------------|--|------------|--|
| Inactive Flag | Group Name | Expression | Level | Operator | Web Template Name | | | |
| N | Tile | | 1 | AND | Edit Tile | | | |
| N | | Desktop | 1 | | | | | |
| N | | EditList | 2 | | | | | |
| N | | Tile | 3 | | | | | |
| N | Map | | 2 | AND | Edit Map | | | |
| N | | Desktop | 1 | | | | | |
| N | | EditList | 2 | | | | | |
| N | | Map | 3 | | | | | |

- g** Configure the manifest for the presentation model for each applet visualization that you defined in [Step 1](#).

You add the UI object, object expressions, and files until the Manifest Administration screen resembles the following configuration.

The screenshot displays the Siebel Manifest Administration interface. It is divided into three main sections: UI Objects, Object Expression, and Files.

UI Objects: This section contains a table with the following data:

| Inactive Flag | Type | Usage Type | Name |
|---------------|--------|--------------------|---------------------|
| N | Applet | Presentation Model | Contact List Applet |

Object Expression: This section contains a table with the following data:

| Inactive Flag | Group Name | Expression | Level | Operator | Web Template Name |
|---------------|------------|------------|-------|----------|-------------------|
| N | Map Pst | | 1 | AND | |
| N | | Desktop | 1 | | |
| N | | Map | 2 | | |

Files: This section contains a table with the following data:

| Inactive Flag | Name |
|---------------|-------------------|
| N | siebelmapmodel.js |

- h** Repeat [Step g](#) for each applet visualization that you configured in Siebel Tools.

- i Configure the physical renderer for each applet visualization that you defined in [Step 1](#).
You add the UI object, object expressions, and files until the Manifest Administration screen resembles the following configuration:

The screenshot shows the Manifest Administration screen in Siebel Tools. It has three main sections: UI Objects, Object Expression, and Files.

UI Objects:

| Inactive Flag | Type | Usage Type | Name |
|---------------|--------|-------------------|---------------------|
| N | Applet | Physical Renderer | Contact List Applet |

Object Expression:

| Inactive Flag | Group Name | Expression | Level | Operator | Web Template Name |
|---------------|------------|------------|-------|----------|-------------------|
| N | Grid | | 1 | AND | |
| N | | Desktop | 1 | | |
| N | | EditList | 2 | | |
| N | | Grid | 3 | | |
| N | Tile | | 2 | AND | |
| N | | Desktop | 1 | | |
| N | | EditList | 2 | | |
| N | | Tile | 3 | | |
| N | Map | | 2 | | |

Files:

| Inactive Flag | Name |
|---------------|-----------------------------|
| N | siebelTilesrollcontainer.js |

If you do not do this administration, then Siebel Open UI uses the jqgridrenderer.js file for the physical renderer for a list applet, by default.

- 3 (Optional) Modify the strings that Siebel Open UI uses for the labels of the applet visualization buttons.

Do the following:

- a In Siebel Tools, choose the Screens application-level menu, click System Administration, and then click List of Values.
- b In the List of Values list, query the Type property for OUI_MODE_VISUALIZATION.

- c Make sure the Language-Independent Code property for each record that Siebel Tools displays in the List of Values list includes the same string that you modified in [Step g](#).

For example, make sure the Language-Independent Code property includes the following values:

| Type | Display Value | Language-Independent Code |
|------------------------|---------------|---------------------------|
| OUI_MODE_VISUALIZATION | Tile | Tile |
| | Map | Map |
| | Grid | Grid |

Siebel Open UI uses the value that the Display Value property contains as the label for each applet visualization button. To view these buttons, see [Figure 31 on page 182](#).

- d Compile your modifications.
- e Log in to the client.
- f Navigate to the Administration - Application screen, and then the Manifest Expressions view.
- g In the Manifest Expressions view, modify the following strings, as necessary.

| Name | Expression |
|------|--------------------------------------|
| Tile | GetObjectAttr("VisualMode") = 'Tile' |
| Map | GetObjectAttr("VisualMode") = 'Map' |
| Grid | GetObjectAttr("VisualMode") = 'Grid' |

For example, Siebel Open UI uses the Tile string in the Expression field for the Tile expression. You can modify these strings to meet your deployment requirements.

- 4 Test your modifications:
 - a Log out of the client, and then log back in.
 - b Navigate to the Contacts screen, and then the Contacts List view.
 - c Verify that Siebel Open UI displays the Grid, Tile, and Map visualization buttons.
The visualization buttons must resemble the buttons that [Figure 31 on page 182](#) displays.
 - d Click each visualization button, and then verify that Siebel Open UI displays the visualization that is associated with the button that you click.

Configuring Manifests for Predefined Visualizations

Table 14 summarizes different manifest configurations for visualizations that come predefined with Siebel Open UI. It includes all the configuration required. For example, you do not configure any expressions or files for web templates.

Table 14. Configuring Manifests for Predefined Visualizations

| Visualization | Presentation Model | Physical Render | Web Template |
|---------------|--|---|---|
| Tile | Set Usage Type to Presentation Model. Set Name to List Applet Name. Add the following to the Files list: siebel / listmodel.js | Set Usage Type to Physical Renderer. Set Name to List Applet Name. Add the following to the Files list: siebel / Tilescrollcontainer.js | Set Usage Type to Web Template. Set the Name to Edit Tile. |
| Grid | Same as Tile. | Set Usage Type to Physical Renderer. Set Name to List Applet Name. Add the following to the Files list: siebel / jqgridrenderer.js | No manifest administration is necessary. You use Siebel Tools to configure Edit List web templates. |
| Map | Same as Tile except add the following file: siebel / mappmodel.js | Same as Grid except add the following file: siebel /custom/ siebelmaprenderer.js | Set Usage Type to Web Template. Set the Name to Edit Tile. |

The following physical renderer modifies the List presentation model so that it can use the Google Map plugin for jQuery:

```
siebel /custom/siebelmaprenderer.js
```

Oracle provides this file only as an example that does a map visualization for a list applet. Oracle does not support usage of siebelmaprenderer.js with Google Maps.

Displaying Applets Differently According to the Applet Mode

This topic describes how to configure Siebel Open UI to display applets differently according to the applet mode. It includes the following topics:

- [“Configuring Siebel Open UI to Use Different Web Templates According to the Applet Mode” on page 190](#)
- [“Configuring Siebel Open UI to Use Different Physical Renderers and Presentation Models According to the Applet Mode” on page 193](#)

The *applet mode* is a type of behavior of an applet web template that determines whether or not the user can or cannot create, edit, query, or delete Siebel CRM records in an applet. Edit, Edit List, Base, New, and Query are examples of applet modes. This topic describes how to modify the presentation model, or to modify the physical render and web templates, to set the applet mode for an applet.

You can use a web template to modify the physical layout of objects in the client that the Siebel Server renders as containers, such as the markup for an applet container. You can also use a physical renderer to modify how the client renders objects in the client, for example, to modify the markup that it uses to display a grid, menu, or tab.

For more information about applet modes and how to configure them in Siebel Tools, see the topic that describes how to control how the user creates, edits, queries, and deletes CRM data in *Configuring Siebel Business Applications*.

Configuring Siebel Open UI to Use Different Web Templates According to the Applet Mode

The example in this topic configures Siebel Open UI to display the same applet differently according to the following responsibility that Siebel CRM assigns to the current user:

- Display the applet as an editable list for the CEO.
- Display the applet as an editable grid for a Business Analyst.

To implement this example, you configure Siebel Open UI to use more than one web template, where each of these web templates reference a different web template file:

- You use the predefined Applet List (Base/EditList) web template that references the CCAppletList_B_EL.swt file. This file uses an editable list layout.
- You add a new Edit Grid List web template that references the EditGridList.swt file. This file uses an editable grid layout.

You configure manifest expressions to determine the web template that Siebel Open UI uses according to the user who is currently using the client.

This example configures the Contact List Applet to include the following applet web templates:

- Edit List applet web template that runs in edit list mode and uses the Applet List(Base/EditList) web template.
- Edit Grid List applet web template that runs in edit list mode and uses the Applet List web template.

To configure Siebel Open UI to use different web templates according to the applet mode

1 Examine the predefined web template that this example uses:

a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

b In the Object Explorer, click Web Template.

c In the Web Templates list, query the Name property for the following value:

"Appl et Li st (Base/Edi tLi st)"

d In the Object Explorer, expand the Web Template tree, and then click Web Template File.

e Notice the value that the Filename property contains.

This example uses the predefined Applet List (Base/EditList) web template to display the applet in a list layout that the user can edit. This web template uses the CCAppl etLi st_B_EL.swt file to display this layout. It is not necessary to modify this web template for this example.

2 Add a custom web template:

a In the Object Explorer, click Web Template.

b In the Web Templates list, add the following web template.

| Property | Value |
|----------|----------------|
| Name | Edit Grid List |

c In the Object Explorer, click Web Template File.

d In the Web Template Files list, add the following web template file.

| Property | Value |
|----------|--|
| Name | Edit Grid List |
| Filename | Specify the file that Siebel Open UI must use to display this applet in a grid layout that the user can edit. For example: Edi tGri dLi st. swt |

3 Modify the applet:

- a** Do [Step 1 on page 182](#), but also add the following applet web template to the Contact List Applet.

| Property | Value |
|--------------|---|
| Name | Edit Grid List |
| Web Template | Edit Grid List You specify the web template that you added in Step 1 . |
| Type | Edit List |

- b** Compile your modifications.

4 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.
- b** Navigate to the Administration - Application screen, and then the Manifest Expressions view.
For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).
- c** In the Expressions list, add the following expressions.

| Name | Expression |
|------------|---|
| Exp_User 1 | GetProfileAttr("Primary Responsibility Name") = "Admin" |
| Exp_User 2 | GetProfileAttr("Primary Responsibility Name") = "CEO" |

For more information, see [“GetProfileAttr Method” on page 483](#).

- d** Navigate to the Manifest Administration view.
- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Web Template |
| Name | Contact List Applet |

- f In the Object Expression list, add expressions until this list resembles the following configuration.

| Object Expression Menu New Delete Query Move Up Move Down Indent >> Outdent << 1 - 8 of 8 | | | | | | |
|---|----------------------|------------|-------|----------|-------------------|--|
| Inactive Flag | Group Name | Expression | Level | Operator | Web Template Name | |
| N | Exp_User1_AppletMode | | 1 | AND | Edit List 1 | |
| N | | Exp_User 1 | 1 | | | |
| N | | EditList | 2 | | | |
| N | Exp_User2_AppletMode | | 2 | AND | Edit Grid List | |
| N | | Exp_User 2 | 1 | | | |
| N | | EditList | 2 | | | |

Note the following:

- You specify the same name that you examined in [Step 1](#) for the Web Template Name for user 1.
- You specify the same name that you added in [Step 2](#) For the Web Template Name for the user 2.
- You specify the expressions that you added in [Step c](#). These expressions configure Siebel Open UI to display an edit list for a user who possesses the CEO responsibility, and a grid for a user who possesses the Business Analyst responsibility.
- If the Usage Type is Web Template, then you do not specify any files in the Files list.

5 Test your modifications:

- a Log in to the client as a user that Siebel CRM associates with the CEO responsibility, and then make sure Siebel Open UI uses the Edit List web template to display the applet as a list.
- b Log out of the client, log back in to the client as a user that Siebel CRM associates with the Business Analyst responsibility, and then make sure Siebel Open UI uses the Edit Grid List web template to display the applet as a grid.

Configuring Siebel Open UI to Use Different Physical Renderers and Presentation Models According to the Applet Mode

The example in this topic configures Siebel Open UI to download different presentation models and physical renderers depending on the following mode that the Contact List Applet must use:

- **Edit List mode.** Download a file named list_PM.js for the custom presentation model and a file named list_PR.js for the custom physical renderer.
- **New mode.** Download a file named new_PM.js for the custom presentation model and a file named new_PR.js for the custom physical renderer.

You can use any name for your custom presentation models and physical renderers.

To configure Siebel Open UI to use different physical renderers and presentation models according to the applet mode

- 1 Customize your presentation models and physical renderers.

In this example, assume you customized the following files:

- list_PM.js
- list_PR.js
- new_PM.js
- new_PR.js

- 2 Add your custom presentation models and physical renderers to the manifest:

- a Log in to the client with administrative privileges.
- b Navigate to the Administration - Application screen, and then the Manifest Files view.

For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- c In the Files list, add the following files that you customized in [Step 1](#).

| Field | Value |
|-------|--------------------------|
| Name | siebel/custom/list_PM.js |
| Name | siebel/custom/list_PR.js |
| Name | siebel/custom/new_PM.js |
| Name | siebel/custom/new_PR.js |

- 3 Configure the manifest for Edit List mode:

- a Navigate to the Manifest Administration view.
- b In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Presentation Model |
| Name | Contact List Applet |

- c In the Object Expression list, add the following expression.

| Field | Value |
|------------|----------|
| Expression | EditList |
| Level | 1 |

- d** In the Files list, add the following file:

`si ebel /custom/l i st_PM. j s`

Siebel Open UI uses the file that you specify for the presentation model that it uses to display the Contact List Applet in Edit List mode.

- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Contact List Applet |

- f** In the Object Expression list, add the following expression.

| Field | Value |
|------------|----------|
| Expression | EditList |
| Level | 1 |

- g** In the Files list, add the following file:

`si ebel /custom/l i st_PR. j s`

Siebel Open UI uses the file that you specify for the physical renderer that it uses to display the Contact List Applet in Edit List mode.

4 Configure the manifest for New mode:

- a** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Presentation Model |
| Name | Contact List Applet |

- b** In the Object Expression list, add the following expression.

| Field | Value |
|------------|-------|
| Expression | New |
| Level | 1 |

- c** In the Files list, add the following file:

```
si ebel /custom/new_PM. j s
```

Siebel Open UI uses the file that you specify for the presentation model that it uses to display the Contact List Applet in New mode.

- d** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|---------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Contact List Applet |

- e** In the Object Expression list, add the following expression.

| Field | Value |
|------------|-------|
| Expression | New |
| Level | 1 |

- f** In the Files list, add the following file:

```
si ebel /custom/new_PR. j s
```

Siebel Open UI uses the file that you specify for the physical renderer that it uses to display the Contact List Applet in New mode.

- 5** Test your modifications.

Adding Custom User Preferences to Applets

This topic describes how to customize default applet behavior so that Siebel Open UI remembers the actions the user takes that effect this behavior. Expand and collapse is an example of this behavior. The example in this topic customizes a physical renderer to display the Opportunity List Applet applet as expanded or collapsed, by default, depending on how the user most recently displayed the applet. For example, assume the user navigates to the Opportunity List Applet, and then expands the applet. Siebel Open UI then displays more records in the list. In the predefined behavior, if the user logs out of the client, logs back in to the client, and then navigates to this list again, then Siebel Open UI does not remember that the user expanded the list. This topic describes how to customize Siebel Open UI so that it remembers this user action. You can use this example as a guideline to modify a predefined applet behavior or to create your own custom applet behavior.

To add custom user preferences to applets

- 1** Add the user preference to your custom physical renderer and presentation model:
 - a** Use a JavaScript editor to open your custom physical renderer that renders the Opportunity List Applet.

- b** Add the custom user preference. You add the following code:

```
var pm = this.GetPM();
var inputPS = CCFMi scUtil _CreatePropSet();
inputPS.SetProperty("Key", "user_preference_name");
inputPS.SetProperty("user_preference_name", "user_preference_value");
pm.OnControlEvent(sieConstants.get("PHYEVENT_INVOKE_CONTROL"),
pm.Get(sieConstants.get("SWE_MTHD_UPDATE_USER_PREF")), inputPS);
pm.SetProperty("user_preference_name", "user_preference_value");
```

- c** Use a JavaScript editor to open your custom presentation model that renders the Opportunity List Applet.
- d** Add a presentation model property that references the custom user preference. You add the following code:

```
var pm = this.GetPM();
var value = pm.Get("user_preference_name");
```

You must make sure that Siebel Open UI derives your custom presentation model from the Presentation Model class. This class contains the logic that saves user preferences in presentation model properties.

- 2** Add the expand and collapse button:

- a** Use a JavaScript editor to open the physical renderer that you edited in [Step a on page 196](#).

- b** Add the following code to the end of the Show method:

```
var id1 = this.GetPM().Get("GetFullId") + '-siebui-cust-expandcollapse-btn';
var expcolbtn = "<button " +
"id= '" + id1 + "' " +
"class= 'appletButton' " +
"aria-label=ExpandCollapse " +
"type=\"button\" " +
"title=ExpandCollapse " + ">" + "ExpandCollapse" + "</button>";
```

- c** Add the following code to the end of the BindEvent method. This code binds the button click.

```
$("#" + pm.Get("GetFullId") + "-" + "siebui-cust-expandcollapse-
btn").bind("click", {ctx: this},
function (e) {
var self = e.data.ctx,
pm = self.GetPM();
SiebelJS.Log("Expand");
var inputPS = CCFMi scUtil _CreatePropSet();
var value = pm.Get ("Expand-Collapse");
inputPS.SetProperty("Key", "Expand-Collapse");
if(value === "Collapse")
{
inputPS.SetProperty("Expand-Collapse", "Expand");
pm.SetProperty("Expand-Collapse", "Expand");
}
else
{
inputPS.SetProperty("Expand-Collapse", "Collapse");
pm.SetProperty("Expand-Collapse", "Collapse");
}
```

```

    }
    pm.OnControlEvent(sieebConsts.get("PHYEVENT_INVOKE_CONTROL"), pm.Get(sieebConsts.get("SWE_MTHD_UPDATE_USER_PREF")), inputPS);
    if(value === "Collapse")
    {
        pm.SetProperty("Expand-Collapse", "Expand");
        //Write Code to expand the applet
        $("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapse-applet-content").show();
    }
    else
    {
        pm.SetProperty("Expand-Collapse", "Collapse");
        //Write Code to collapse the applet
        $("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapse-applet-content").hide();
    }
}
);

```

- d** Add the following code to the end of the ShowUI method. This code accesses the default value of the custom Expand-Collapse user preference, and then instructs Siebel Open UI to display the applet as expanded or collapsed according to the user preference value:

```

PhysicalRenderer.prototype.ShowUI()
{
    var pm = this.GetPM();
    var value = pm.Get("Expand-Collapse");
    if(value === "Collapse")
    {
        //Write Code to collapse the applet
        $("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapse-applet-content").hide();
    }
    else
    {
        //Write Code to expand the applet
        $("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapse-applet-content").show();
    }
}

```

- e** Use an HTML editor to open the HTML that Siebel Open UI uses to display the Opportunity List Applet, and then add the following code:

```

$("#s_" + this.GetPM().Get("GetFullId") + "_div").find(".siebui-collapse-applet").append(expcolbtn);

```

For more information about how to edit HTML code for an applet, see [“Customizing Client Logo, Background, and Style” on page 145](#).

3 Test your modifications:

- a** Log in to the client, and then navigate to the Opportunity List Applet.
- b** Click the expand and collapse button, and then verify that Siebel Open UI expands the applet.

- c Log out of the client, log back in to the client, navigate to the Opportunity List Applet, and then verify that Siebel Open UI displays the same expanded state that you set in [Step b](#).

Customizing Applets to Capture Signatures

A *signature capture* is an electronic capture of a user's signature. Siebel Open UI stores this signature as an image. It eliminates the need for paper storage and retrieval. It is recommended that you configure Siebel Open UI to store each signature capture for the Siebel Open UI client and for the high-interactivity client as an image that Siebel CRM can use in the administrative view and the reports view. This configuration simplifies how Siebel Open UI renders the signature capture across these user interfaces. Siebel Open UI can then get the signature capture from Siebel BI Publisher for reporting purposes and process it, as required.

Siebel Open UI clients and high-interactivity clients use different formats to store each signature capture. This topic describes how to configure Siebel Open UI so that it can display these formats simultaneously from the CIC control and to retrieve them in a report. For more information about:

- Siebel BI Publisher, see *Siebel Reports Guide*
- The CIC control, see Article ID 869586.1 on My Oracle Support

To customize applets to capture signatures

- 1 Copy a signature form applet that comes predefined with Siebel Open UI:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b In the Object Explorer, click Applet.

- c In the Applets list, locate an applet that includes a signature capture configuration.

For this example, locate the following applet:

LS Pharma Cal I Signature Form Applet

This topic uses Siebel Pharma as an example. You can modify the objects for your Siebel application, as necessary.

- d Right-click the applet you located in [Step c](#), and then click Copy Record.

- e Add an _OUI suffix to the name.

For example:

LS Pharma Cal I Signature Form Applet_OUI

- 2 Add applet user properties:

- a In the Object Explorer, expand the Applet tree, and then click Applet User Prop.

- b** In the Applet User Props list, add the following applet user properties.

| Name | Value |
|---------------------------------|-------|
| CanInvokeMethod: ClearSignature | TRUE |
| Signature Min Length | 5 |

- 3** Add controls:

- a** In the Object Explorer, click Control.
- b** In the Controls list, add the following controls.

| Name | Description |
|-----------------------|--|
| Clear Signature | Set the MethodInvoked property to ClearSignature. |
| Address | Set the Field property to Address. |
| Signature Capture | Set the following properties: <ul style="list-style-type: none"> ■ Set the Field property to Signature. ■ Set the HTML Type property to InkData. |
| Disclaimer Text | Set the Read Only property to TRUE. |
| Signature Header Text | |

- 4** Add an applet web template:

- a** In the Object Explorer, click Applet Web Template.
- b** In the Applet Web Templates list, right-click the Base applet web template, and then click Copy Record.
- c** Set the following properties.

| Property | Value |
|----------|-------|
| Name | Edit |
| Type | Edit |

- 5** Modify the drilldown objects:

- a** In the Object Explorer, click Drilldown Object.
- b** In the Drilldown Objects list, modify the following value of the Hyperlink Field property of the Apply Drilldown and the Cancel Drilldown drilldown objects.

| Old Value | New Value |
|-----------------------|-----------|
| Signature Header Text | Address |

6 Copy a predefined view:

- a** In the Object Explorer, click View.
- b** In the Views list, locate a view that includes a signature capture configuration.

For this example, locate the following view:

LS Pharma Cal I Si gnature Capture Vi ew

- c** Right-click the view you located in [Step b](#), and then click Copy Record.
- d** Add an _OUI suffix to the name.

For example:

LS Pharma Cal I Si gnature Capture Vi ew_OUI

7 Modify the view web template:

- a** In the Object Explorer, expand the View tree, expand the View Web Template tree, and then click View Web Template Item.
- b** In the View Web Template Items list, query the Name property for the following value:

LS Pharma Cal I Si gnature Form Appl et

- c** Modify the following value of the Name property.

| Old Value | New Value |
|--------------------------------------|--|
| LS Pharma Call Signature Form Applet | LS Pharma Call Signature Form Applet_OUI |

- d** Modify the following value of the Applet Mode property.

| Old Value | New Value |
|-----------|-----------|
| Base | Edit |

8 Modify a call form applet that comes predefined with Siebel Open UI:

- a** In the Object Explorer, click Applet.
- b** In the Applets list, locate an applet that includes a call form configuration.

For this example, locate the following applet:

Pharma Professi onal Cal I Form Appl et

- c** In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
- d** In the Applet User Props list, add the following applet user property.

| Name | Value |
|-----------------------|--|
| Signature Applet Name | LS Pharma Call Signature Form Applet_OUI |

- e In the Object Explorer, click Drilldown Object.
- f In the Drilldown Objects list, query the Name property for Signature Capture Drilldown.
- g Modify the following value of the View property.

| Old Value | New Value |
|---------------------------------------|---|
| LS Pharma Call Signature Capture View | LS Pharma Call Signature Capture View_OUI |

- 9 Modify the screen:
- a In the Object Explorer, click Screen.
 - b In the Screens list, locate a screen that displays the signature form and call form applets.
For this example, locate the following screen:

LS Pharma Call Screen

- c In the Object Explorer, expand the Screen tree, and then click Screen View.
- d In the Screen Views list, query the Name property for the following value:
LS Pharma Call Signature Capture View
- e Modify the following value of the View property.

| Old Value | New Value |
|---------------------------------------|---|
| LS Pharma Call Signature Capture View | LS Pharma Call Signature Capture View_OUI |

- 10 Compile your modifications.

- 11 Configure your customization:

- a Log in to the client with administrative privileges.
- b Navigate to the Administration - Application screen, and then the Views view.
- c In the Views list, query the Name property for the following value:
LS Pharma Call Signature Capture View
- d Make a note of the field values of the responsibility that the client displays in the Responsibilities list.
- e In the Views list, add the following view.

| Field | Value |
|-----------|---|
| View Name | LS Pharma Call Signature Capture View_OUI |

- f In the Responsibilities list, add a responsibility. Use the same field values that you noted in [Step d](#).
- g Navigate to the Administration - Personalization screen, and then the Applets view.

- h** In the Applets list, add the following applet.

| Field | Value |
|-------|--|
| Name | LS Pharma Call Signature Form Applet_OUI |

- i** In the Rule Sets list, add the following rule set.

| Field | Value |
|------------|--|
| Name | Pharma Call Default |
| Sequence | 1 |
| Start Date | Any date that has already occurred. For example, 01/01/2012. |

- 12** Make sure you configure Siebel Open UI to store each image on a secure system using disk encryption or system encryption.
- 13** Test your modifications.

Customizing Fields

This topic describes how to customize fields. It includes the following information:

- [Displaying and Hiding Fields on page 203](#)
- [Configuring Spell Checker on Fields on page 204](#)

Displaying and Hiding Fields

The example in this topic describes how to configure Siebel Open UI to display a field. To view a diagram that illustrates some of the objects you modify and the relationships between these objects, see [“Configuring Manifests” on page 128](#).

This topic is similar to the [“Refreshing Applets That Contain Modified Data” on page 159](#) topic, but with fewer details. It demonstrates how you can quickly modify a presentation model.

To display and hide fields

- 1** Log in to the Siebel application, navigate to the Contact Screen, and then the Contact List view.
- 2** Drill down on the first contact, and then navigate to the third level Affiliations view.
- 3** Click New, and then add a new contact.
- 4** Download a copy of the partialrefreshpm.js file to the following folder on the client computer:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

The code in this file includes the logic that Siebel Open UI uses to display or hide a field. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. For more information about this file, see [“Example of a Presentation Model” on page 42](#).

5 Configure the manifest.

Do [Step 2 on page 160](#), except in [Step f on page 160](#) set the Expression to Desktop.

For more information, see [“Configuring Manifests” on page 128](#).

6 Test your modifications:

- a Clear the browser cache.

For more information, see [“Clearing the Browser Cache” on page 204](#).

- b Navigate to the Contact list.

- c Drill down on a contact, and then click the Affiliations tab in the third-level navigation.

- d Remove all information from the Job Title field.

- e Make sure Siebel Open UI removes the Work Number and Fax Number fields from the user interface.

Clearing the Browser Cache

The example in this topic describes how to clear the browser cache in Google Chrome. The steps you do might vary slightly for other browsers or various releases of Google Chrome.

To clear the browser cache

- 1 Open Google Chrome.
- 2 Click the wrench icon.
- 3 Click Settings, History, and then click Clear All Browsing Data.
- 4 In the Clear Browsing Data dialog box, do the following:
 - a Choose The Beginning of Time in the dropdown list.
 - b Make sure all options include a check mark.
 - c Click Clear Browsing Data.

Configuring Spell Checker on Fields

The example in this topic describes how to configure Spell Checker for the Summary field of the Service Request form applet.

To configure spell checker on fields

- 1 Log in to the client.
- 2 Navigate to the Service Request screen, and then the Service Request list.
- 3 Right-click in the Summary field of the service request form, and then click Inspect Element.

Siebel Open UI displays the HTML source code for the applet in a separate Developer Tools window with the following code highlighted. This code defines the Summary field:

```
<textarea name="s_1_1_128_0" aria-labelledby="Abstract_Label " aria-label="Summary" style="height: 96px; width: 192px" maxlength="100" aria-disabled="false" value=""></textarea>
```

For more information about using Developer Tools, see [“Customizing Client Logo, Background, and Style” on page 145](#).

- 4 Replace the code highlighted in [Step 3](#) with the following code:

```
<input type="text" name="s_1_1_128_0" value="" aria-labelledby="Abstract_Label " aria-label="Summary" style="height: 96px; width: 192px; " maxlength="100" aria-disabled="false">
```

Siebel Open UI comes predefined to check the spelling of any text the user enters in a text field. To enable spell checking on a textarea field, you must modify this field to a text field.

7

Customizing Calendars and Schedulers

This chapter describes how to customize calendars and schedulers. It includes the following topics:

- [Customizing Calendars on page 207](#)
- [Customizing Resource Schedulers on page 215](#)

Customizing Calendars

This topic includes examples of customizing the calendar that Siebel Open UI displays. It includes the following information:

- [Deploying Calendars According to Your Calendar Deployment Requirements on page 208](#)
- [Using Fields to Customize Event Styles for the Calendar on page 208](#)
- [Customizing Event Styles for the Calendar on page 211](#)
- [Customizing Repeating Calendar Events for Siebel Mobile on page 212](#)
- [Controlling How Calendars Display Timestamps on page 213](#)
- [Replacing Standard Interactivity Calendars on page 214](#)

Deploying Calendars According to Your Calendar Deployment Requirements

Table 15 describes different deployment requirements and the work you must do to meet each requirement. To deploy calendars according to your calendar deployment requirements, locate the requirement in the Deployment Requirement column, and then do the work that the Work You Must Do column describes.

Table 15. Deployment Requirements and the Work You Must Do to Meet Each Requirement

| Deployment Requirement | Work You Must Do |
|---|---|
| <p>Your deployment requires the following items:</p> <ul style="list-style-type: none"> ■ Requires standard interactivity calendars ■ Does not require new calendar features that the 2012 Innovation Pack introduces | <p>Do not modify the repository.</p> <p>Siebel Open UI adds no new calendar features that the 2012 Innovation Pack introduces, such as the Event Style or Workweek features. All calendars work correctly:</p> <ul style="list-style-type: none"> ■ High interactivity calendars work in Siebel Open UI the same way that they work in the high-interactivity client while in high-interactivity mode. ■ Standard interactivity calendars do not work in Siebel Open UI. They will continue to work in the same way that they previously worked in the high and standard interactivity clients. ■ Siebel Open UI calendars work in Open UI mode. |
| <p>Your deployment requires the following items:</p> <ul style="list-style-type: none"> ■ Requires Siebel Open UI Calendars ■ Requires new features that the 2012 Innovation Pack introduces ■ Does not require standard interactivity calendars | <p>Import your Siebel Open UI modifications and a separate SIF file.</p> <p>This SIF file imports Siebel Open UI modifications and a separate file that updates standard interactivity calendars to use the new Open UI control. If the user runs the Siebel application in high interactivity or standard-interactivity mode, then Siebel Open UI does not render any standard interactivity calendar and it might create an error.</p> <p>Oracle includes this SIF file in a ZIP file in the Tool s\RepPatch folder. For more information, see <i>Siebel Maintenance Release Guide</i> on My Oracle Support.</p> |

Using Fields to Customize Event Styles for the Calendar

Siebel Open UI comes predefined to use the Status field in the Action business component to supply the event style, by default. You can modify it to use any bounded, single-value field that resides in the Action business component.

To use fields to customize event styles for the calendar

- 1** Identify the applet that you must modify:
 - a** In the client, navigate to the calendar page that displays the style that you must modify.
 - b** Click the Help menu, and then click About View.
 - c** Copy the applet name that the dialog box displays to the clipboard.
- 2** Identify the field that must supply the event style:
 - a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click Business Component.
 - c** In the Business Components list, locate the Action business component.
 - d** In the Object Explorer, expand the Business Component tree, and then click Field.
 - e** In the Fields list, identify a bounded, single-value field.
Siebel Open UI will use this field to supply the values that it displays in the Legend in the calendar in the client.
- 3** Modify the applet:
 - a** In the Object Explorer, click Applet.
 - b** Click in the Applets list, click the Query menu, and then click New Query.
 - c** Paste the applet name that you copied in [Step 1](#) into the Name property, and then press the Enter key.
To modify styles for:
 - ❑ **All calendar applets.** You can add user properties to the HI Calendar Base Applet. Siebel Open UI uses this applet to set styles for all applets.
 - ❑ **One specific applet.** You can add user properties to an individual applet. User properties that you define on an individual applet override the styles that the HI Calendar Base Applet specifies.
 - d** In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
 - e** In the Applet User Props list, add the following applet user property.

| Property | Value |
|----------|---|
| Name | CSS Event Style |
| Value | Enter the name of the field that you identified in Step 2 . |

- f** In the Applet User Props list, add the following applet user property.

| Property | Value |
|----------|---|
| Name | CSS Event Style LOV |
| Value | Enter the LOV type that the field that you identified in Step 2 uses. |

Siebel Open UI will use the values that this list of values contains to populate the CSS Class tags in the HTML, and then to render the event and legend styles. It uses the EventStyle property that contains the language independent code. It uses the set of language independent codes that this field contains to define the range of possible values. The CSS Event Style LOV user property allows you to define a single set of styles that Siebel Open UI can use for all languages in a multilingual environment.

If the CSS Event Style user property does not exist, or if the CSS Event Style LOV user property does not exist, then Siebel Open UI uses the following default values:

- Status for the field.
- EVENT_STATUS for the list of values.

- 4** Compile your modifications.
- 5** Restart the Siebel application.
- 6** In the client, navigate to the Administration - Data screen, and then click List of Values.
- 7** Query the List of Values list for all of the unique language independent codes that exist for this list of values type.
For example, query the Type field for TODO_TYPE.
- 8** Use a style sheet editor to open the theme-calendar.css file.
- 9** For each value that you find in [Step 7](#), create the following two styles.

| Style | Description |
|--|---|
| .fc-event-skin.calendar-EventStyle-LOVName | Siebel Open UI uses this style for the event. |
| #color_square_LOVName | Siebel Open UI uses this style for the square that it displays on the legend. |

When Siebel Open UI creates the HTML to render the Calendar, it specifies these styles in the CLASS tag for the event and for the legend. It specifies the strings for the language independent code for the field with spaces removed. For example:

- ? .fc-event-skin.calendar-EventStyle-Completed and #color_square_Completed
- ? .fc-event-skin.calendar-EventStyle-NotStarted and #color_square_NotStarted

? .fc-event-skin.calendar-EventStyle-InProgress and #color_square_InProgress

For an example of customizing a style sheet, see [“Customizing Event Styles for the Calendar” on page 211](#).

10 Save the theme-calendar.css file.

11 Clear the browser cache.

For more information, see [“Clearing the Browser Cache” on page 204](#).

12 Navigate to the Calendar view.

13 Make sure Siebel Open UI displays the correct styles.

Customizing Event Styles for the Calendar

Style sheet attributes determine the color, transparency, font, and other styles for each status. You can modify these styles. You can use any single value field that resides in the Action business component to determine the style that Siebel Open UI uses to render events in the calendar. Siebel Open UI uses the value that the Status field contains to determine how the client displays an event in the calendar, by default. For example:

- Done
- Not Started
- Planned
- Success

To customize event styles for the calendar

- 1** Use an editor to open the theme-calendar.css file.
- 2** Locate the code that specifies the style that you must modify.

For example:

```
#color_square_LOVName {color: #d3ffd7!important;}
.fc-event-skin.calendar-EventStyle-LOVName {
  {custom_attributes}
```

where:

- LOVName identifies the event status that you must modify, such as Done or Not Started.
 - *custom_attributes* specify the style properties you can modify, such as the background color or font type.
- 3** Modify the code, as necessary.

For example:

```
#color_square_Done {color: #d3ffd7!important;}
.fc-event-skin.calendar-EventStyle-Done {
  background: #d3ffd7;
  border-color: #A8FFAF;
}
```

In this example, Siebel Open UI modifies the style for each Done appointment. It also modifies the style for the Done entry in the legend that it displays in the upper-left corner of the calendar.

If Siebel Open UI cannot find a matching style for a LOVName, then it displays events in the default text color, which is typically black on white.

- 4 Save your modifications, clear the browser cache, and then verify that Siebel Open UI displays the style you defined for the Done status.

For more information, see ["Clearing the Browser Cache" on page 204](#).

Customizing Repeating Calendar Events for Siebel Mobile

Siebel Open UI allows the user to specify values for the Workdays field and the Week Start field. It uses the user preferences that reference the Locale values, by default. It stores the following items:

- Stores locale preferences in the Locale table (S_LOCALE).
- Stores user preferences as predefault values from Locale values.
- Stores user preferences in the user preferences file.

Specifying Work Days

If the user sets the user preference for the Weekly Calendar View to Work Week, then Siebel Open UI displays only the days that are specified as workdays. This preference can be specified at several levels, so Siebel Open UI uses the following priority:

- 1 Personal user preference.
- 2 Locale preference for the current user locale.
- 3 Applet user property. This property provides high interactivity support.
- 4 If none of these items are set, then Siebel Open UI displays the Monday through Friday, five day workweek.

Specifying the First Day of the Week

If the set of visible days does not include the First Day of Week preference, then Siebel Open UI displays the next visible day. For example, if the user uses a Monday through Friday, five-day workweek, and if the First Day of Week is Saturday, then Siebel Open UI displays Monday as the first day of the week in the Work Week calendar. It does this because Monday is the first visible day that occurs after Saturday.

Specifying Work Days and the First Day of the Week

You can define a default value for all users according to the locale, but a user can override this value. For example, assume the following:

- The existing Work Week setting for all users is Monday through Friday, as determined by the Locale settings that the Siebel administrator sets.
- A set of users work Monday through Friday.
- Another set of users who provide weekend support work Wednesday through Sunday.
- Each weekend user logs into the Siebel client and uses the User Preferences Calendar view to set their Wednesday through Sunday schedule. Siebel Open UI stores this modification in the user preferences file.

In this situation, Siebel Open UI does the following:

- Displays Monday through Friday for each user who does not use the User Preferences Calendar view to modify their preference
- Displays Wednesday through Sunday for each user who uses the User Preferences Calendar view to modify their preference

Controlling How Calendars Display Timestamps

You specify an applet user property to control how the calendar displays timestamps.

To control how calendars display timestamps

- 1 Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- 2 In the Object Explorer, click Applet.
- 3 In the Applets list, locate any calendar applet.
- 4 In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
- 5 In the Applet User Props list, query the Name property for the following value:

Enable Daily Time Display - for Daily View

- 6 Set the Value property to one of the following values:
 - **Always.** Always display the timestamp immediately before the meeting subject. For example, 8:00 AM - 9:00 AM My Meeting.
 - **Never.** Do not display the timestamp.
 - **Off-interval.** Display the timestamp immediately before the meeting subject only if the meeting starts or ends at a time that is not consistent with the user preference that specifies how to display time intervals. For example, if the user preference includes intervals of 8:00, 8:30, 9:00, and so on, and if a meeting occurs from:
 - **8:00 to 8:30.** Do not display the timestamp.

❑ **8:03 to 8:14.** Display the timestamp.

❑ **8:00 to 8:15.** Display the timestamp.

An *off-interval meeting* is a meeting that does not start and end on a calendar increment. For example, if the calendar displays 30 minute increments, and if the user creates a meeting that does not start and end on the half-hour, then this meeting is an off-interval meeting. A 15 minute meeting that starts at 9:05 AM is an example of an off-interval meeting.

If you do not specify an applet user property for a:

■ **Daily view or weekly view.** Siebel Open UI uses an off-interval value.

■ **Monthly view.** Siebel Open UI always displays the timestamp.

7 Repeat [Step 5](#) and [Step 6](#) for the following applet user property:

Enable 5Day Time Display - for weekly view

8 Repeat [Step 5](#) and [Step 6](#) for the following applet user property:

Enable Monthly Time Display - for Monthly View

Replacing Standard Interactivity Calendars

Some standard interactivity calendars do not work properly in Siebel Open UI. This topic describes how to replace the calendars that standard-interactivity uses with the calendars that Siebel Open UI uses.

To replace standard interactivity calendars

1 Modify the applet:

a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

b In the Object Explorer, click Applet.

c In the Applets list, query the Name property for the following value:

LS CIM eCalendar Weekly Applet

d Modify the Class property from C\$SSWEFrameCalGridLS to the following value:

C\$SSWEFrameActHl Cal Grid

This modification replaces standard-interactivity applets.

e Compile your modifications.

2 Test your modifications:

a Log in to the client.

- b** Make sure Siebel Open UI displays the correct applets.

For example, make sure the Fullcalendar applet replaces the LS CIM eCalendar Weekly Applet.

Customizing Resource Schedulers

This topic describes how to customize a resource scheduler. It includes the following topics:

- ["Overview of Customizing Resource Schedulers" on page 216](#)
- ["Customizing a Resource Scheduler" on page 217](#)
- ["Customizing the Filter Pane in Resource Schedulers" on page 229](#)
- ["Customizing the Resource Pane in Resource Schedulers" on page 231](#)
- ["Customizing the Timescale Pane in Resource Schedulers" on page 234](#)
- ["Customizing the Schedule Pane in Resource Schedulers" on page 241](#)
- ["Customizing Tooltips in Resource Schedulers" on page 249](#)

This topic includes example values that customize the resource scheduler that Siebel Hospitality uses. You can use a different set of values to customize a different Siebel application.

Overview of Customizing Resource Schedulers

Figure 32 includes an example of a *resource scheduler*, which is a type of bar chart that includes a schedule that allows the user to schedule a resource. In this example, the Function Space Diary is a resource scheduler that allows the user to schedule a room in a hotel. The room is the resource. You can use a different resource scheduler to meet the deployment requirements of your Siebel application.

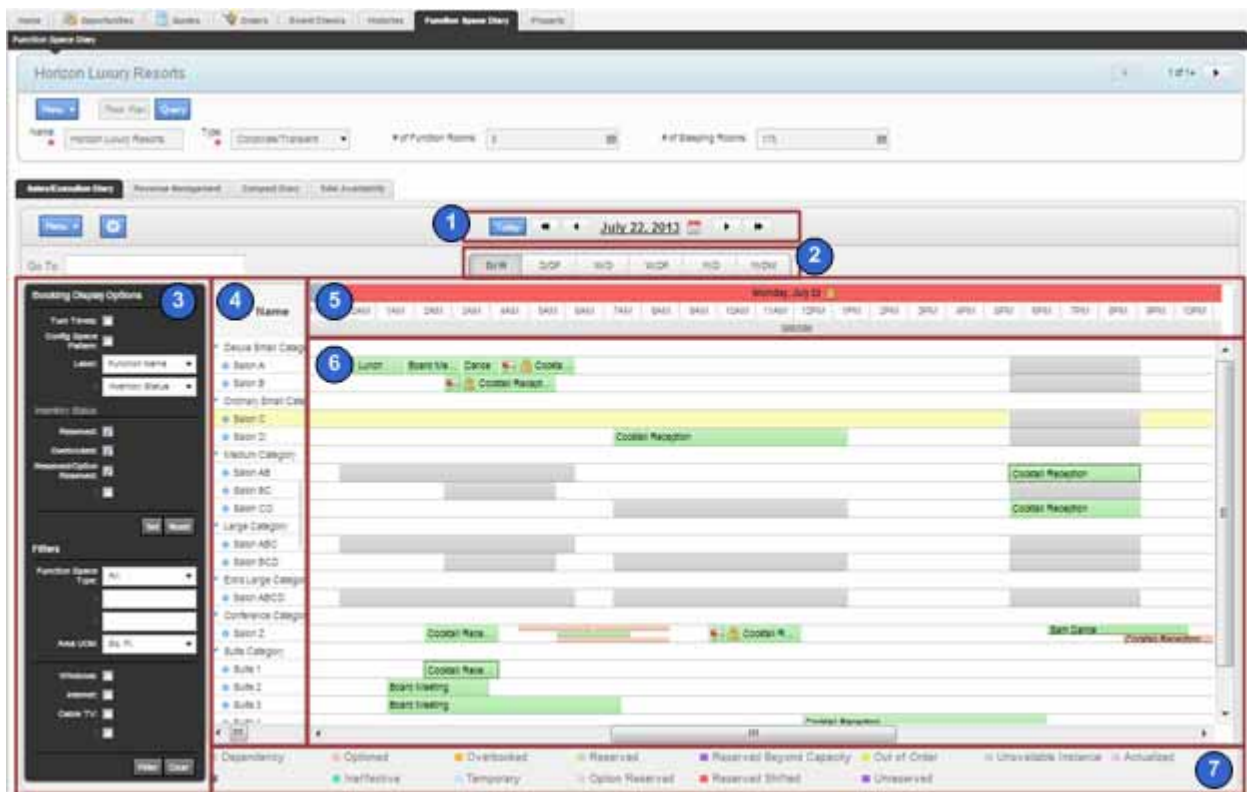


Figure 32. Example of a Resource Scheduler

Explanation of Callouts

The resource scheduler includes the following items:

- 1 **Date navigation bar.** Allows the user to modify the date that **Siebel Open UI** displays in the schedule.
- 2 **Time scale selector.** Includes the following time scales:
 - **D/H.** Days and hours.
 - **D/DP.** Days and day parts.
 - **W/D.** Weeks and days.
 - **W/DP.** Weeks, days, and day parts.

- **M/D.** Months and days.
- **M/DW.** Months, days of the week, and day parts.

A *day part* is a time period that occurs during the day. For example, morning, afternoon, evening, and night are examples of day parts. You can customize the time period that defines a day part. For example, the morning day part comes predefined as 8:00 AM to Noon. You can modify it to another time period, such as 9:00 AM to Noon. For information about customizing the day part, see [Step 5 on page 235](#).

- 3 Filter pane.** Allows the user to filter data that Siebel Open UI displays in the schedule.
- 4 Resource pane.** Displays a list of resources. A *resource* is something that a resource scheduler can use to support an event. A room is an example of a resource. An *event* is something that occurs in a resource. A meeting is an example of an event.
- 5 Timescale pane.** Displays a time scale that includes date and time information. It includes the following items:
 - The *major axis* is a dimension that Siebel Open UI displays in the time scale. In this example, the major axis displays the current day, which is Monday, July 22.
 - The *minor axis* is a dimension that Siebel Open UI displays in the time scale. In this example, the minor axis displays the time of day, such as 10:00 AM.
 - The *third axis* is a dimension that Siebel Open UI displays in the time scale. It displays this axis as a third dimension in addition to the major axis and the minor axis. You can use the third axis to display Siebel CRM information according to your deployment requirements. In this example, the third axis displays the total number of rooms that are available for the current day. For example, 300/380 indicates that 300 rooms out of a total of 380 rooms are available for the current day.
- 6 Schedule pane.** Displays the schedule as a timeline. Includes events that are scheduled for each resource.
- 7 Legend.** Displays a legend that describes the meaning of each color that Siebel Open UI displays in the Schedule pane.

Using Abbreviations When Customizing the Resource Scheduler

An *abbreviation* is an optional shortened version of a value that you can specify in the Value property of an object that a resource scheduler uses. ST is an example of an abbreviation. It indicates the start time of a resource scheduler. Siebel Open UI uses these abbreviations to reduce the amount of information that it sends from the Siebel Server to the client. This book includes the abbreviations that you can use for Siebel Hospitality. Unless noted elsewhere, these abbreviations come predefined with Siebel Open UI, and you can use only the abbreviations that this book describes. For help with using abbreviations, see [“Getting Help from Oracle” on page 35](#).

Customizing a Resource Scheduler

This topic describes how to customize a resource scheduler.

To customize a resource scheduler

1 Configure the applet that Siebel Open UI uses to display the resource scheduler:

a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

b In the Object Explorer, click Applet.

This example includes the minimum set of objects that you must add. To view predefined applets that Siebel Open UI uses for a resource scheduler, you can query the name property for TNT Function Bookings Gantt Applet or FS DB Planned GanttChart AX Applet. To simplify creating your resource scheduler, you can make a copy of one of these applets, and then modify the copy.

c In the Applets list, add a new applet, or copy one of the applets mentioned in [Step b](#).

d Set the following property for the applet that you added in [Step c](#).

| Property | Value |
|----------|------------------|
| Class | CSSSWEFrameGantt |

e In the Object Explorer, expand the Applet tree, and then click Applet User Prop.

f In the Applet User Props list, add the following applet user properties. Each value in the Value property supports this example. You can use values that your deployment requires. You must include all of these user properties.

| Name | Value | Description |
|-----------------------|--|--|
| Gantt Open UI Service | TNT Gantt UI Service | Specify the business service name that Siebel Open UI uses to save system preferences and user preferences. |
| Physical_Renderer | GanttTNTRenderer | Specify the name of the class that Siebel Open UI uses for the physical renderer. |
| Presentation_Model | GanttTNTPresentationModel | Specify the name of the class that Siebel Open UI uses for the presentation model. |
| ClientPMUserProp | EnableTooltip, Date Padding for TimeScale LIC, DateBar Navigation TS | Specify the user properties that Siebel Open UI makes available to JavaScript files that reside on the client. You must use a comma to separate each user property name. |

| Name | Value | Description |
|--------------------------------|--|--|
| Date Padding for TimeScale LIC | <i>time_scale_identifier</i> : <i>number_of_pages</i> | Specify the number of pages that Siebel Open UI uses in the cache for the time scale. For more information, see “Customizing the Cache That Siebel Open UI Uses for Time Scales” on page 227 . |
| DateBar Navigation TS | <i>time_scale_identifier</i> : <i>small_date_change</i> , <i>big_date_change</i> | Specify the date navigation buttons. For more information, see “Customizing the Date Navigation Buttons” on page 227 . |
| Duration for TimeScale LIC | <i>time_scale_identifier</i> : <i>number_of_days</i> For example: 1: 7; 2: 1; 4: 1; 32: 36; 64: 31; 128: 7; 256: 35; 512: 1; 1024: 1 | Specify the number of days that Siebel Open UI sends to the cache for each time scale. For example, the following value specifies to send seven days of data to the cache for the Week/Day time scale: 1: 7 You can use a semicolon to specify days for more than one time scale. Siebel Open UI uses a number to identify each time scale. For more information, see “Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229 . |
| No. Of Panes | 3 | This applet user property specifies the number of panes that Siebel Open UI displays. A resource scheduler always displays the Resource pane, Time Scale pane, and the Scheduler pane, so you must not modify this applet user property. |
| Custom Control Name | s_Diary | Specify the name of the custom control. |

| Name | Value | Description |
|------------------|-------------------------------------|---|
| Custom Control | "Legend Control Name,s_Legend" | Specify the tag that Siebel Open UI uses to render each custom control. Siebel Open UI uses this information to parse the input property set when it renders a custom control. Use the following format for each value: <i>control_name, tag_name</i> where: ■ <i>control_name</i> specifies the name of the custom control. ■ <i>tag_name</i> specifies the tag name that you define in the Tag Name control user property in Step c on page 225 . For example, the following value specifies to use the s_Legend tag for the Legend Control Name control: Legend Control Name, s_Legend You can use Custom Control 1 and Custom Control 2 to specify more controls, as required. |
| Custom Control 1 | "DateBar Control Name,s_DateBar" | |
| Custom Control 2 | "GanttChart,s_Diary" | |

g Configure system preferences and user preferences.

In the Applet User Props list, add the following applet user properties. Each value in the Value property supports this example. You can use values that your deployment requires. You must include all of these user properties.

| Name | Value | Description |
|---|---|--|
| Support System Preferences | Y | If the value is N or empty, then Siebel Open UI does not support system preference usage with a resource scheduler. |
| Support User Preferences | Y | |
| System_Pref Field <i>number</i> For example, System_Pref Field 1, System_Pref Field 2, and so on. | "GntAXCtrl:Time Scale", "TST", "TNT_SHM_GNTAX_TIME_SCALE" | <p>Specify the default values that Siebel Open UI uses in a resource scheduler. You can use the following abbreviations:</p> <ul style="list-style-type: none"> ■ TST. Specifies the Time Scale. ■ ST. Specifies the start time of the schedule. ■ ET. Specifies the End time of the schedule. <p>If a field is a LOV field, then you must specify the LOV name so that the code gets the language-dependent value.</p> <p>For more information about the abbreviations that the Value property contains, see "Using Abbreviations When Customizing the Resource Scheduler" on page 217.</p> |
| System_Pref_Prefix | GntAXCtrl: | Specify the prefix that Siebel Open UI uses for every system preference that it uses with a resource scheduler. You must use this prefix. Siebel Open UI only queries system preferences that include this prefix. It does this query in the System Preferences business component. |

| Name | Value | Description |
|--|--|---|
| User_Pref Field <i>number</i> For example, User_Pref Field 1, User_Pref Field 2, and so on. | "Display Toggle - Query", "Display Toggle" | Specify the user preference name. Siebel Open UI sends an abbreviation of this user preference to the client. |
| User_Pref_Prefix | Diary | Specify the prefix that Siebel Open UI uses for every user preference that it uses with a resource scheduler. You must use this prefix. Siebel Open UI queries only the user preferences that include this prefix. It does this query in the User Preferences business component. |

You can use these system preferences and user preferences to configure Siebel Open UI to do decision making in your custom JavaScript code that resides on the client. For example, you can set a user preference for the default time scale to Month and Day, and then use this default in your custom JavaScript code to set the default time scale. User preferences take precedence over system preferences. If a user preference exists, then Siebel Open UI uses it instead of the corresponding system preference.

- h Specify the methods that Siebel Open UI uses with the Siebel Server.

In the Applet User Props list, add the following applet user properties. These applet user properties specify the methods that Siebel Open UI uses with the Siebel Server. You must add them so that Siebel Open UI can call the methods that reside on the Siebel Server. You must not modify these methods. You must also add a CanInvokeMethod applet user property for every method that your custom JavaScript calls on the Siebel Server. Make sure you set the Value property for each of these applet user properties to True.

| Property | Description |
|--|---|
| CanInvokeMethod: DoInvokeDrilldown | A resource scheduler supports drilldowns through the Resource, Schedule, and Timescale panes. If the user clicks a label in one of these panes, then Siebel Open UI calls the DoInvokeDrilldown method. |
| CanInvokeMethod: DoOperation | Calls the DoOperation method. Siebel Open UI calls this method for various events, such as drag and drop, extend, shrink, create task, and so on. |
| CanInvokeMethod: FilterDisplayOptions | Specifies how Siebel Open UI displays bookings when the user clicks Set to set criteria in the Filter pane. You must configure Siebel Open UI to call the FilterDisplayOptions method, typically through the Set button. This configuration enables Siebel Open UI to filter events according to the attributes that it defines for each control. |

| Property | Description |
|--------------------------------------|--|
| CanInvokeMethod: FilterGantt | Specifies how Siebel Open UI displays bookings when the user sets a criteria in the Filter pane. |
| CanInvokeMethod: InitPopup | Calls the popup dialog box for some operations, such as drag and drop, create task, and so. You cannot customize this behavior. |
| CanInvokeMethod: InvokeOperation | Specifies the method that Siebel Open UI calls when the user clicks a button in the popup applet. You cannot customize this behavior. This popup applet is the TNT Gantt Popup Applet that Siebel Open UI configures for the applet user property. |
| CanInvokeMethod: ReSetFilterGantt | Resets the resource filter options to default values. Siebel Open UI displays these options in the Filter pane. |
| CanInvokeMethod: RefreshGantt | Calls the Refreshgantt method. Siebel Open UI uses this method to refresh a resource scheduler. |
| CanInvokeMethod: ResetDisplayOptions | Resets the display filter options to default values. Siebel Open UI displays these options in the Filter pane. |
| CanInvokeMethod: SaveControlValues | Stores the user preference values that the Filter pane fields contain. You cannot customize this behavior. |

2 Configure optional applet user properties.

You use applet user properties for most of the optional customization that you can configure in a resource scheduler. For more information about how to do this customization, see the following topics:

- [“Customizing the Filter Pane in Resource Schedulers” on page 229](#)
- [“Customizing the Resource Pane in Resource Schedulers” on page 231](#)
- [“Customizing the Timescale Pane in Resource Schedulers” on page 234](#)
- [“Customizing the Schedule Pane in Resource Schedulers” on page 241](#)
- [“Customizing Tooltips in Resource Schedulers” on page 249](#)

3 Add controls:

- a In the Object Explorer, click Control.

- b** In the Controls list, add the following controls.

| Name | Caption - String Reference |
|------|------------------------------|
| 1 | SBL_TNT_TS_WEEK_DAY |
| 2 | SBL_TNT_TS_DAY_DAYPART |
| 4 | SBL_TNT_TS_DAY_HOUR |
| 64 | SBL_TNT_TS_MONTH_DAY |
| 128 | SBL_TNT_TS_WEEKDAY_DAYPART |
| 256 | SBL_TNT_TS_MONTH_DAY_OF_WEEK |

Note the following:

- ❑ A resource scheduler requires each of these controls for the time scale.
- ❑ You must add a control for each time scale.
- ❑ Set the Name property of each control to the time_scale_identifier, such as 1, 2, 4, and so on. Siebel Open UI uses a number to identify each time scale, such as 128 or 256. It does not use values 8, 16, or 32 for time scales with Siebel Hospitality. It might use different values for a different Siebel application. For more information, see [“Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229](#).
- ❑ Set the HTML Type property of each control to MiniButton.
- ❑ Set the Method Invoked property of each control to RefreshGantt.

- c** In the Controls list, add the following controls.

| Name | HTML Type | Class | Description |
|--------------|---------------|------------------|---|
| GanttChart | CustomControl | CSSSWEFrameGantt | Specifies the main resource scheduler control. |
| GanttDateBar | CustomControl | CSSSWEFrameGantt | Specifies the Date bar that contains the date controls. Allows the user to modify the date in a resource scheduler. |
| Legend | CustomControl | CSSSWEFrameGantt | Specifies the legend that Siebel Open UI displays in a resource scheduler. |
| GoToResource | Field | Leave empty. | Specifies the optional input text control that searches for resources that reside in the Resource pane. Siebel Open UI binds the event to this control in the JavaScript that resides on the client, so you must use GoToResource as the name. |

Make sure you set the Caption - String Reference property of the GoToResource control to SBL_GO_TO-1004233041-4MM. Do not set this property for the other controls.

- d** In the Object Explorer, expand the Control tree, click Control User Prop, and then use the Control User Props list to add the following control user properties to each of the controls that you added in [Step c](#).

| Parent Control | Value Property |
|----------------|----------------|
| GanttChart | s_Diary |
| GanttDateBar | s_DateBar |
| Legend | s_Legend |

Set the Name property for each control user property to Tag Name. Each of these control user properties specifies a tag name for the control. This configuration allows the JavaScript code to access the tag.

- 4** Edit the web template:
- a** In the Object Explorer, click Applet Web Template.

- b** In the Applet Web Templates list, create the following applet web template.

| Property | Value |
|------------------|------------------|
| Name | Edit |
| Type | Edit |
| Web Template | Applet OUI Gantt |
| Upgrade Behavior | Admin |

- c** In the Object Explorer, click Applet.
- d** In the Applets list, right-click the applet that you are modifying, and then click Edit Web Template.
- e** In the Web Template Editor, add each of the controls that you added in [Step 3](#) and [Step c](#) to the layout.
- It is recommended that you position each of these controls on the right side of the layout.
- f** Set the Item Identifier property of the GanttDateBar control to 3000.
- g** Close the Web Layout Editor.

5 Configure the application:

- a** In the Object Explorer, click Application.
- b** In the Applications list, query the Name property for the application that you are modifying.
- c** In the Object Explorer, expand the Application tree, and then click Application User Prop.
- d** In the Application User Props list, add the following application user property.

| Property | Value |
|----------|---|
| Name | ClientBusinessService <i>number</i> For example, ClientBusinessService1. |
| Value | Gantt UI Service |

You must add a new application user property for each business service that your customization calls in the client. In this example, you specify the Gantt UI Service business service. You must increment the Name for each application user property that you add. For example, ClientBusinessService1, ClientBusinessService2, and so on.

- 6** Compile your modifications.
- 7** Test your modifications:
- a** Log in to the client.
- b** Navigate to the resource scheduler, and then test your modifications.

Customizing the Cache That Siebel Open UI Uses for Time Scales

This topic describes how to customize the cache that Siebel Open UI uses for time scales.

To customize the cache that Siebel Open UI uses for time scales

- 1 Specify the number of pages to use in the cache for a time scale.

Use the following value for the Date Padding for TimeScale LIC applet user property:

time_scale_identifier.number_of_pages

where:

- *time_scale_identifier* specifies the time scale.
- *number_of_pages* specifies the number of pages that Siebel Open UI uses for the previous operation and for the next operation. It uses these pages when it prepares the page cache for the time scale that the *time_scale_identifier* specifies.

The following example specifies the Week/Day time scale LIC, and it specifies to use 2 pages for the previous operation, and 2 pages for the next operation:

1: 2

Siebel Open UI uses a number to identify each time scale. It uses the number 1 to identify the Week/Day time scale. For more information, see [“Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229](#).

Siebel Open UI always includes a default page, so it uses the following calculation to determine the total cache page count:

previous pages + default page + next pages

So, the cache size for the 1:2 example is 5:

$2 + 1 + 2 = 5$

For more examples:

- **1:1.** Use three pages (1+1+1).
 - **1:0.** Use one pages (0+1+0).
 - **1:2.** Use five pages (2+1+2).
- 2 (Optional) Add more than one time scale.

Use a semicolon to separate each time scale. For example:

1: 1; 2: 1; 4: 1; 32: 1; 64: 1; 128: 1; 256: 1; 512: 1; 1024: 1

Customizing the Date Navigation Buttons

When you specify the DateBar Navigation TS applet user property, you specify the time period that that Siebel Open UI uses to reset the current date when the user clicks one of the following buttons:

- **Left arrow.** Displays the previous date, small date change.

- **Right arrow.** Displays the next date, small date change.
- **Double left arrow.** Displays the previous date, large date change.
- **Double right arrow.** Displays the next date, large date change.

Siebel Open UI displays these buttons to the left and to the right of the date that it displays in the Date Navigation bar.

To customize the date navigation buttons

- 1 Specify the DateBar Navigation TS applet user property.

Use the following format:

time_scale_identifier: small_date_change, big_date_change

where:

- *time_scale_identifier* identifies the time scale. Siebel Open UI uses a number to identify each time scale. For more information, see [“Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229](#).
- *small_date_change* specifies the number of hours, days, weeks, or months that Siebel Open UI uses to modify the current date if the user clicks the left arrow or the right arrow.
- *big_date_change* specifies the number of hours, days, weeks, or months that Siebel Open UI uses to modify the current date if the user clicks the double left arrow or the double right arrow.

- 2 (Optional) Add more than one time scale.

Use a semicolon to separate each time scale. For example:

1: 7, 30; 4: 1, 7; 2: 1, 7; 64: 30, 365; 128: 7, 30; 256: 1, 35;

Examples of Customizing Date Navigation Buttons

The following value customizes the date navigation buttons:

1: 7, 30

where:

- **1.** Specifies the *time_scale_identifier*. For example, 1 specifies the Week/Day time scale.
- **7.** Specifies the number of days. For example, if the current date is August 15, 2013, and if the user clicks:
 - The left arrow, then Siebel Open UI displays August 8, 2013 as the current date.
 - The right arrow, then Siebel Open UI displays August 22, 2013 as the current date.
- **30.** Specifies the number of days for the record set. For example, if the current date is August 15, 2013, and if the user clicks:
 - The double left arrow, then Siebel Open UI displays July 15, 2013 as the current date.

- The double right arrow, then Siebel Open UI displays September 15, 2013 as the current date.

For another example:

4: 1, 7

where:

- **4.** Specifies the time_scale_identifier. For example, 4 specifies the Day/Hour time scale.
- **1.** Specifies the number of days. For example, if the current date is August 15, 2013, and if the user clicks:
 - The left arrow, then Siebel Open UI displays August 14, 2013 as the current date.
 - The right arrow, then Siebel Open UI displays August 16, 2013 as the current date.
- **7.** Specifies the number of days for the record set. For example, if the current date is August 15, 2013, and if the user clicks:
 - The double left arrow button, then Siebel Open UI displays August 8, 2013 as the current date.
 - The double right arrow, then Siebel Open UI displays August 22, 2013 as the current date.

Determining the Number That Siebel Open UI Uses to Identify Time Scales

This topic describes how to determine the number that Siebel Open UI uses to identify a time scale.

To determine the number that Siebel Open UI uses to identify time scales

- 1 Log in to the Siebel client with administrative privileges.
- 2 Navigate to the Administration - Data screen, and then the List of Values view.
- 3 Query the Type field for the following value:

TNT_SHM_GNTAX_TIME_SCALE

- 4 In the Display Value field, locate the time scale that you must modify.
- 5 In the Language-Independent Code field, make a note of the value.

Siebel Open UI uses the number that it displays in the Language-Independent Code field to identify the time scale that it displays in the Display Value field.

Customizing the Filter Pane in Resource Schedulers

You can add a custom filter that controls how Siebel Open UI filters resources and determines the label colors that it uses for events. You add these controls in the Filter pane. For example, you can add a filter control named Type to filter events according to the value that the Type field contains.

To customize the Filter pane in resource schedulers

- 1** In the Object List Editor, choose the applet that you modified in [Step 1 on page 218](#).
- 2** In the Object List Editor, expand the Applet tree, and then click Control.
- 3** (Optional) Configure the resource scheduler to filter resources:
 - a** In the Controls list, choose a control that meets your deployment requirements that Siebel Open UI can use to filter resources.
 If no existing controls meet your deployment requirements, then you can add a control.
 - b** In the Object List Editor, expand the Control tree, and then click Control User Prop.
 - c** In the Control User Props list, add the following control user property.

| Name | Value | Description |
|------------|---------------------|--|
| Field Name | Max Room Area Sq Ft | Specify to use the control as part of the resources filter. The HTML Type property of this control must be set to Text so that Siebel Open UI displays a text box that allows the user to enter a value. Siebel Open UI then uses the filter resources according to the value that the user enters. For example, if the user enters a value of 100, then Siebel Open UI sends the following value to the FilterGantt business service method. It sends this value as an input argument: Max Room Area Sq Ft = "100" |

- 4** (Optional) Configure the resource scheduler to filter resources and events:
 - a** In the Controls list, choose a control that meets your deployment requirements that Siebel Open UI can use to filter resources and events.
 If no existing controls meet your deployment requirements, then you can add a control.
 - b** In the Object List Editor, expand the Control tree, and then click Control User Prop.

- c** In the Control User Props list, add the following control user property.

| Name | Value | Description |
|--------------------|----------|---|
| Display Field Name | Optioned | Specify to use the control as part of the resources filter. The HTML Type property of this control must be set to CheckBox so that Siebel Open UI displays a checkbox that allows the user to display Optioned events. Siebel Open UI then filters resources and events according to the choice that the user makes. In this example, if the user adds a check mark, then Siebel Open UI sends the following value to the DisplayOptions business service method. It sends this value as an input argument: Optioned = "Y" |

- 5** Use the Web Layout Editor to add the control that you modified in [Step 3](#) or [Step 4](#) to the Filter pane in the web template.

You can do this work as part of [Step e on page 226](#).

Customizing the Resource Pane in Resource Schedulers

This topic describes how to customize the Resource pane.

To customize the Resource pane in resource schedulers

- 1** In the Object List Editor, choose the applet that you modified in [Step 1 on page 218](#).
- 2** In the Object List Editor, expand the Applet tree, and then click Applet User Prop.
- 3** In the Applet User Props list, add each of the following applet user properties, as required.

| Name | Value | Description |
|--------------------------|----------|--|
| Pane 0 Grid Name | Resource | Specify the name of the Resource pane. |
| Pane 0 Grid Type | RGrid | Specify the pane type. |
| Pane 0 Col <i>number</i> | NM,Name | Specify the details for the column header that Siebel Open UI displays in the Resource pane, including the abbreviated name and the label. |

| Name | Value | Description |
|---|---------------|---|
| Pane 0 Col <i>column number</i> Attr <i>column attribute number</i> For example: Pane 0 Col 1 Attr 2 | IID, 206 | Specify the identifier that identifies the icon that Siebel Open UI displays for the column. |
| Pane 0 Col 0 Attr 1 | FLD,Room Name | Specify the Room Name business component field that Siebel Open UI uses to get the value, and then display it under resource column 0. |
| Pane 0 Col 0 Attr 2 | IDD,Products | Specify the following items: <ul style="list-style-type: none"> ■ IDD. The abbreviation that indicates the name of the drill down object. ■ Drilldown field. The business component field that Siebel Open UI uses when the user drills down to a destination view. <p>If the user clicks the DDFLD value that Siebel Open UI displays under resource column 0, then it navigates the user to the view that the Products drill down object defines.</p> <p>Siebel Open UI uses the Pane 0 Col 0 Attr 2 applet user property in conjunction with the Pane 0 Col 0 Attr 3 applet user property.</p> <p>You must configure the corresponding drilldown object that identifies the destination view and the ID. This drilldown object resides in the applet that you are configuring.</p> |
| Pane 0 Col 0 Attr 3 | DDFLD,Room Id | Specify the following items: <ul style="list-style-type: none"> ■ DDFLD. The abbreviation that indicates the name of the drill down field. ■ Drilldown field. The name of the business component field that Siebel Open UI uses when the user drills down on resource column 0. Siebel Open UI uses this field value to navigate the user to the destination view. <p>Siebel Open UI uses the Pane 0 Col 0 Attr 3 applet user property in conjunction with the Pane 0 Col 0 Attr 2 applet user property.</p> |

| Name | Value | Description |
|----------------------------|-----------------|--|
| Pane 0 Field <i>number</i> | Room Id | Specify the business component field that Siebel Open UI uses to get the Siebel CRM data that it displays in the Resource pane. |
| Pane 0 Join Field | Room Id | Specify the field that Siebel Open UI uses to join resources and events. Resources and events are independent of each other. This join field joins the events that are related to a resource. For example, a meeting is an example of an event that can be held in a room, which is an example of a resource. In this example, each event includes a Room Id. |
| Pane 0 Parent Field | Parent Room Id | Specify the parent business component field that Siebel Open UI uses to display resources in a hierarchy. |
| Pane 0 Start Date Field | Effective Start | Specify the Start Date field that Siebel Open UI uses to prepare a search specification. |
| Pane 0 View Mode | 3 | <p>Specify the view mode that this Resource pane supports. You must use the following numbers to indicate each view mode:</p> <ul style="list-style-type: none"> ■ 0. VIEW_SALESREP. ■ 1. VIEW_MANAGER. ■ 2. VIEW_PERSONAL. ■ 3. VIEW_ALL. ■ 4. VIEW_NONE. ■ 5. VIEW_ORG. ■ 6. VIEW_CONTACT. ■ 7. VIEW_GROUP. ■ 8. VIEW_CATALOG. ■ 9. VIEW_SUBORG. <p>You can use a comma to specify more than one view mode, where the comma separates each number. For example, 1,2,3.</p> |

- 4 Configure the font color that Siebel Open UI uses in the Resource pane.

Add the following applet user property.

| Property | Value | Description |
|--------------------|--------|--|
| Pane 0 Color Field | Status | Specify the business component field that determines the color that Siebel Open UI uses to display a resource. If you do not specify a value, then Siebel Open UI displays only the color black. |

- 5 Configure the icons that Siebel Open UI display next to the Resource Name label in the Resource pane.

Add the following applet user property.

| Property | Value |
|---------------------------|--|
| Pane 0 Icon <i>number</i> | Specify the name of a field that Siebel Open UI displays in the Resource pane, a comma, and then the CSS class that contains the icon. For example: Room Backup Requi red, si ebui -backuprequi red |

Customizing the Timescale Pane in Resource Schedulers

This topic describes how to customize the Timescale pane.

To customize the Timescale pane in resource schedulers

- 1 In the Object List Editor, choose the applet that you modified in [Step 1 on page 218](#).
- 2 In the Object List Editor, expand the Applet tree, and then click Applet User Prop.
- 3 In the Applet User Props list, add each of the following applet user properties, as required.

| Name | Value | Description |
|-----------------------|---|---|
| Pane 1 Grid Name | TimeScale | Specify the name of the Timescale pane. |
| Pane 1 Grid Type | TGrid | Specify the type of the Timescale pane. |
| Pane 1 BC Name | TNT SHM Property Special Dates Action | Specify the business component name that Siebel Open UI uses to get information about special days or events that it displays in the Timescale pane. It can use this information to display colors and icons on the Timescale pane. |
| Pane 1 End Date Field | End Date | Specify the End Date business component field where Siebel Open UI applies a search specification to prepare special days, events information, and so on. |

| Name | Value | Description |
|----------------------------|------------------------------|--|
| Pane 1 Start Date Field | Start Date | Specify the Start Date business component field where Siebel Open UI applies a search specification to prepare special days, events information, and so on. |
| Pane 1 Field <i>number</i> | Start Date,SD | Specify the name of a field that resides in the Data business component. Siebel Open UI requires an abbreviation to prepare special day information. Siebel Open UI sends the field value as an abbreviation to the client so that the client JavaScript files can use this information. |
| Time Scale LOV | TNT_SHM_GNTA X_TIME_SCALE | Specify the LOV name that Siebel Open UI uses for different time scales. |

- 4 Configure the third axis that Siebel Open UI displays on the Timescale pane. Add each of the following user properties, as required.

| Name | Value | Description |
|---|---|--|
| Pane 1 BottomAxis Date Field | Start Date | Specify the Date field that Siebel Open UI uses to search the third axis that resides in the TimeScale pane business component. |
| Pane 1 BottomAxis Field <i>number</i> where <i>number</i> is a field number. | Total Group Available,FLD1 | Specify the third axis that resides in the TimeScale pane business component field. The value contains the name and abbreviation as FLD1, FLD2, and so on. |
| Pane 1 BottomAxisBC Name | TNT SHM FSI Auth Lvl for Calendar | Specify the business component name that Siebel Open UI uses to get the data that it displays in the third axis. If you do not include this applet user property, then Siebel Open UI does not display the third axis in the Timescale pane. |
| Pane 1 BottomAxisBC Search Spec | [Product Type] = LookupValue(PR ODUCT_TYPE, 'Sleeping Room') | Specify the search specification that Siebel Open UI applies on the business component for the Third axis in the TimeScale pane. |
| Pane 1 BottomAxisBC Sort Spec | Start Date | Specify the sort specification that Siebel Open UI applies on the business component for the Third axis in the TimeScale pane. |

- 5 Configure the Day Part time scale:

- a** Add the Day Part time scale button to the controls.

Use 2 for the Name of this button. This configuration is the LIC value that Siebel Open UI uses for the Day Part time scale. You must use a value from the time scale list of values to name each time scale button control. For more information about how to add this button, see [Step b on page 224](#).

- b** Add each of the following applet user properties, as required.

| Name | Value | Description |
|--|--|---|
| Pane 1 Daypart <i>number</i> where <i>number</i> is the day part number. | Morning,NM,06: 00:00,ST,12:00 :00,ET,21600,D UR | Siebel Open UI uses the following business component to provide the dynamic day part data: TNT SHM Property Day Part Pricing If this business component does not exist, or if it does not contain any records, then Siebel Open UI uses this applet user property to specify the Static Day Part information that the day part time scale uses. The value contains the Name, Starttime, Endtime, and Duration of the daypart. |
| Pane 1 Daypart Field <i>number</i> | Name,NM | Specify the business component fields that Siebel Open UI uses to get the day part information. The value includes the field name and the abbreviation for this field name. |
| Pane 1 DaypartBC Name | TNT SHM Property Day Part Pricing | Specify the name of the business component that Siebel Open UI uses to get the day part information. |
| Pane 1 DaypartBC Search Spec | (Empty) | Specify the search specification that resides on the business component that Siebel Open UI uses to get the day part information. This value comes predefined as empty. |
| Pane 1 DaypartBC Sort Spec | Start Time | Specify the sort specification that resides on the business component that Siebel Open UI uses to get the day part information. |

- 6 Configure the colors that Siebel Open UI displays on the time scale. You can configure Siebel Open UI to modify the colors it uses in time scale cells according to a condition. For example, it can set the color of a weekend cell to red. Add each of the following applet user properties, as required.

| Name | Value | Description |
|--|--|---|
| Pane 1 Color: Admin BC | TNT SHM Gantt AX Admin Function Status | Specify the business component that Siebel Open UI uses to display colors for time scale data cells. |
| Pane 1 Color: Admin BO | TNT SHM Gantt Admin System Pref | Specify the business object that references the business component that Siebel Open UI uses to display colors for time scale data cells. |
| Pane 1 Color Application | Y | Specify how to get the time scale color. You can use one of the following values: <ul style="list-style-type: none"> ■ Y. Get the time scale color from the application object. ■ N. Get the time scale color from an applet user property. |
| Pane 1 Color Type: Color | Holiday: #3ED143 | If the value of the Pane 1 Color Application applet user property is N, then the value of the Pane 1 Color Type: Color applet user property must specify the event and the color that Siebel Open UI uses to indicate this event. In this example, the event is Holiday and the color code is #3ED143. For more information about these color codes, see the ColorHexa website at http://www.colorhexa.com . |
| Pane 1 Color Type: Color <i>number</i> | Special Events: #F76161 | If the value of the Pane 1 Color Application applet user property is N, then the value of the Pane 1 Color Type: Color <i>number</i> applet user property must specify a special event and the color that Siebel Open UI uses to indicate this event. |
| Pane 1 Colors BC Color Field | Color LIC | If the value of the Pane 1 Color Application applet user property is Y, then the value of the Pane 1 Colors BC Color Field applet user property must specify the Color field that resides in the business component that the Pane 1 Color: Admin BC applet user property specifies. |
| Pane 1 Colors BC Type Field | Inventory Status | If the value of the Pane 1 Color Application applet user property is Y, then the value of the Pane 1 Colors BC Type Field applet user property must specify the type of field that resides in business component that the Pane 1 Color: Admin BC applet user property specifies. |

| Name | Value | Description |
|--|---------------------|---|
| Pane 1 Hour Axis Business Service Method | EventsTSHourMap | Specify the business service method that Siebel Open UI uses to get the hour axis colors that it displays in the Timescale pane. |
| Pane 1 Hour Axis Business Service Name | TNT Utility Service | Specify the business service that Siebel Open UI uses to get the hour axis colors that it displays in the Timescale pane. |
| Pane 1 Hour Axis Color | Y | Specify how to color the hour cells. You can use one of the following values: <ul style="list-style-type: none"> ■ Y. Use a variety of colors in the cells. ■ N. Use only black in the cells. |

7 Specify how to display weekends. Add the following applet user properties, as required.

| Name | Value | Description |
|--------------------------------------|-----------------------|---|
| Pane 1 Weekend Application | Y | Specify how to get the weekend information. You can use one of the following values: <ul style="list-style-type: none"> ■ Y. Get the weekend information from the application object. ■ N. Get the weekend information from an applet user property. |
| Pane 1 Weekend BC | TNT SHM Weekend Admin | If the value of the Pane 1 Weekend Application applet user property is Y, then the Pane 1 Weekend BC applet user property must specify the business component that Siebel Open UI uses to get the weekend information. |
| Pane 1 Weekend BC Field:Day | Week Day Num | If the value of the Pane 1 Weekend Application applet user property is Y, then the Pane 1 Weekend BC Field:Day applet user property must specify the business component field that Siebel Open UI uses to get the weekend information. |
| Pane 1 Weekend BC Field:Weekend Flag | Weekend Weekday Flag | If the value of the Pane 1 Weekend Application user property is Y, then the Pane 1 Weekend BC Field:Weekend Flag user property must specify the business component field that Siebel Open UI uses to get the weekend information. The Pane 1 Weekend BC Field:Day user property specifies the day information. The Pane 1 Weekend BC Field:Weekend Flag user property specifies to configure this day as a weekday or as a weekend day. |

| Name | Value | Description |
|--|-------------|---|
| Pane 1 Weekend BO | SHM Site | If the value of the Pane 1 Weekend Application applet user property is Y, then the Pane 1 Weekend BO applet user property must specify the business object that Siebel Open UI uses to get the weekend information. |
| Pane 1 Weekend Property Admin BC | SHM Site | Specify the business component that Siebel Open UI uses to get weekend information from the Siebel Server. |
| Pane 1 Weekend Property Admin BC Field | Property Id | Specify the field that resides in the property business component. |
| Pane 1 Weekend Property BC | SHM Site | Specify the business component that Siebel Open UI uses to get the property information. |
| Pane 1 Weekend Property Field | Property Id | Specify the business component field that Siebel Open UI uses to get the property information. |
| Pane 1 Weekends | 0,5,6 | <p>Specify the days that Siebel Open UI uses as weekend days. If the value of the Pane 1 Weekend Application applet user property is N, then the Pane 1 Weekends applet user property must specify the days that Siebel Open UI uses to identify weekend days. You must use the following numbers to represent each day:</p> <ul style="list-style-type: none"> ■ 0. Sunday. ■ 1. Monday. ■ 2. Tuesday. ■ 3. Wednesday. ■ 4. Thursday. ■ 5. Friday. ■ 6. Saturday. <p>Use a comma to separate each number. For example, a value of 0,5,6 in the Pane 1 Weekends user property customizes Siebel Open UI to use Sunday, Friday, and Saturday as weekend days.</p> |

- 8 Configure the icons that Siebel Open UI displays and the text that it uses with these icons in time scale cells according to a condition. Add the following applet user properties, as required.

| Name | Value | Description |
|------------------------------|------------------------------------|---|
| Pane 1 Icon <i>number</i> | Sell Notes,siebui- sellnotes | Specify the field value from the business component that the Pane 1 BC Name applet user property identifies, and the class name of the cascading style sheet that Siebel Open UI uses to render the time scale cells. You must use a comma to separate these values. You can configure more than one Pane 1 Icon <i>number</i> applet user property. For example, you can configure Pane 1 Icon 1, Pane 1 Icon 2, and so on. |

- 9 Configure the drilldowns that Siebel Open UI uses on the major axis and the third axis. If you configure a drilldown, then you must configure each of the following applet user properties.

| Name | Value | Description |
|--------------------------------|----------------------------|--|
| Pane 1 Date Drilldown | <i>source: destination</i> | Specify the time scale that Siebel Open UI displays when the user clicks a date in the Timescale pane. For more information, see “Customizing Time Scales That Siebel Open UI Displays in the Timescale Pane” on page 240. |
| Pane 1 Item Drilldown Name | Time Scale Drilldown | Specify the drilldown object that resides in the applet that Siebel Open UI uses to display the third axis. You must also configure this drilldown object in the applet. |
| Pane 1 Item Drilldown Field | OUI Property Id | Specify the field that contains the value that Siebel Open UI uses when it does a drill down operation on a label that resides in the third axis. Siebel Open UI uses this field value to navigate the user to the destination view according to the drilldown object that the Pane 1 Item Drilldown Name applet user property specifies. |

Customizing Time Scales That Siebel Open UI Displays in the Timescale Pane

This topic describes how to specify the Pane 1 Date Drilldown applet user property. You specify the time scales that Siebel Open UI displays when the user clicks a date in the Timescale pane, such as Monday, July 22.

To customize time scales that Siebel Open UI displays in the Timescale pane

- 1 Determine the number that Siebel Open UI uses to identify the time scale that you must modify.

For more information, see [“Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229](#).

- 2 Add the value that you determined in [Step 1](#) to the value of the Pane 1 Date Drilldown applet user property. Use the following format:

source: destination

where:

- *source* identifies the time scale that the user clicks. Siebel Open UI uses a number to identify each time scale. For more information, see [“Determining the Number That Siebel Open UI Uses to Identify Time Scales” on page 229](#).
- *destination* identifies the time scale that the resource scheduler displays when the user clicks the source.

For example, the following value configures Siebel Open UI to display the Day/Day-Part time scale when the user clicks the Week/Day time scale:

1: 2

- 3 (Optional) Allow the user to navigate between time scales.

You can use a semicolon to separate each time scale. For example:

1: 2; 2: 256; 4: 256; 64: 2; 128: 2; 256: 2;

Customizing the Schedule Pane in Resource Schedulers

This topic describes how to customize the Schedule pane.

To customize the Schedule pane in resource schedulers

- 1 In the Object List Editor, choose the applet that you modified in [Step 1 on page 218](#).
- 2 In the Object List Editor, expand the Applet tree, and then click Applet User Prop.
- 3 In the Applet User Props list, add each of the following applet user properties, as required.

| Name | Value | Description |
|------------------|-------------|------------------------|
| Pane 2 Grid Name | Utilization | Specify the pane name. |
| Pane 2 Grid Type | UGrid | Specify the pane type. |

| Name | Value | Description |
|-------------------------------|--|--|
| Pane 2 Field <i>number</i> | Function Space Id,FSI | Specify the business component fields that contain the information that Siebel Open UI displays in the Schedule pane. Siebel Open UI sends information from these fields to the client. Use the following format: <i>field name, abbreviated name</i> You can specify more than one field. For example, Pane 2 Field 1, Pane 2 Field 2, and so on. |
| Pane 2 BC Name | TNT SHM Function Booking VBC | Specify the business component that Siebel Open UI uses to get information about the events that it displays in the Schedule pane. |
| Pane 2 BC Sort Spec | Function Space Id, Start Date Time | Specify the business component fields that Siebel Open UI uses for the sort specification that it uses to sort the records that it displays in the Schedule pane. You must use a comma to separate each field name. |
| Pane 2 BC Search Spec | "[Activity Type] = Completed" | Specify the business component fields that Siebel Open UI uses for the search specification that it uses to identify the records that it displays in the Schedule pane. You can use an equation or a field name. For more information about specifying a search specification, see <i>Configuring Siebel Business Applications</i> . |
| Pane 2 End Date Field | Absolute End Date Time | Specify the end date field where Siebel Open UI does the search according to the search specification. |
| Pane 2 Start Date Field | Start Date Time | Specify the start date field where Siebel Open UI does the search according to the search specification. To formulate the search specification, Siebel Open UI joins the field that you specify in the Pane 2 Start Date Field applet user property with the field that you specify in the Pane 2 End Date Field applet user property. |
| Pane 2 Start Attrib | ST | Specify the abbreviation that Siebel Open UI uses for the start date field. |
| Pane 2 End Attrib | ET | Specify the abbreviation that Siebel Open UI uses for the end date field. |
| Pane 2 Join Field | Function Space Id | Specify the field that Siebel Open UI uses as the identifier when it matches rows with other panes. |

| Name | Value | Description |
|---|------------------------|--|
| Pane 2 Bypass Overlap For Status | Dependency | <p>Specify the type of events that Siebel Open UI does not split when an event overlap occurs. An <i>event overlap</i> is a condition that occurs if more than one event occurs at the same time. Siebel Open UI splits the row height of each overlapping event so that it can display them in the same screen space that it normally uses to display an event that does not overlap.</p> <p>In this example, Siebel Open UI does not split any Dependency events that overlap.</p> <p>You can use a comma to bypass multiple event types. For example, you can use the following value to bypass Dependency and Optioned events:</p> <p>Dependency, Optioned</p> |
| Pane 2 Overlap Event LOV Type | TNT_SHM_INV_S TATUS | Specify the LOV type that Siebel Open UI uses for the inventory status when events overlap. |
| Pane 2 Overlap Event Logical Order Based Field Attr | GS | Specify the abbreviation that Siebel Open UI uses for the field that it displays in the Schedule pane when events overlap. |

| Name | Value | Description |
|---|---|--|
| Pane 2 Overlap Event Logical Order Values | Reserved,Option Reserved,Overbooked,Optioned,Unreserved,Unavailable,Unavailable Instance,Out of Order,Temporary | <p>Specify the order that Siebel Open UI uses to display overlapping events, according to status. In this example, Siebel Open UI displays statuses in the following order. It displays Reserved events first and Temporary events last:</p> <ul style="list-style-type: none"> ■ Reserved ■ Option Reserved ■ Overbooked ■ Optioned ■ Unreserved ■ Unavailable ■ Unavailable Instance ■ Out of Order ■ Temporary |
| Pane 2 Round Minutes Events | 15 | <p>Specify the number that Siebel Open UI uses to resize an event. If the user resizes an event, then Siebel Open UI rounds the time according to the value that you specify. For example, assume you specify 15 as the value for this applet user property. Assume an event starts at 08:00 AM and ends 10:00 AM. If the user drags the end time for this event from 10:00 AM to 10:12 AM, then Siebel Open UI rounds this end time according to the closest 15 minute increment, where 15 is measured from the beginning of the hour. In this example, it rounds the end time to 10:15 AM.</p> |

- 4 Configure the colors that Siebel Open UI uses for the events that it displays in the Schedule pane. It modifies these colors according to a condition. For example, it can use red to color a Reserved event. Add each of the following applet user properties, as required.

| Name | Value | Description |
|-----------------------------------|----------------------------|---|
| Pane 2 Color <i>number</i> | INV_STATUS_Reserved, GREEN | Specify the INV_STATUS color that Siebel Open UI uses for the LOV type. |
| Pane 2 Event Color Service Method | EventsColorMap | Specify the business service method that Siebel Open UI uses to get the event colors. |
| Pane 2 Event Color Service Name | TNT Utility Service | Specify the business service that Siebel Open UI uses to get the event colors. |
| Pane 2 Event Default Color | #6495ed | Specify the default color that Siebel Open UI uses for events. |

| Name | Value | Description |
|--|--------------------------|--|
| Pane 2 Status LIC Field <i>number</i> | INVENTORY_STA TUS,GS | Specify the colors that Siebel Open UI uses for the inventory status. For example, specify the abbreviation that you defined in the Pane 2 Overlap Event Logical Order Based Field Attr applet user property. You defined these user properties in Step 3 on page 241 . |
| Pane 2 Status LOV Type | TNT_FSD_COLO R_SCHEMA | <p>Specify the color scheme that Siebel Open UI uses for events. To modify schemes, do the following:</p> <ul style="list-style-type: none"> ■ Log in to the Siebel client with administrative privileges. ■ Navigate to the Administration - Data screen, and then the List Of Values view. ■ Query the Type Field for TNT_FSD_COLOR_SCHEMA. ■ Modify the fields, as necessary. <p>Pane 2 Status LOV Type specifies only the color schemes that are available. To configure Siebel Open UI to display a color according to a condition in Siebel Hospitality, you must use the Function Status Color Schema list that resides in the Function Space Diary Administration view of the Function Space Administration screen. For example, to use red for the Prospect status in Siebel Hospitality. Configuration for your Siebel application might be different than it is for Siebel Hospitality.</p> |

- 5 Configure the icons and the text for these icons that Siebel Open UI uses with the events that it displays in the Schedule pane according to a condition. Add each of the following applet user properties, as required.

| Name | Value | Description |
|------------------------------|--------------------------------------|--|
| Pane 2 Icon <i>number</i> | DNMF,siebui-donotmove | <p>Specify the abbreviation that you defined in the corresponding applet user property and the class where the corresponding cascading style sheet resides. For example, specify the abbreviation that you defined in the Pane 2 Field 0 applet user property. You defined these user properties in Step 3 on page 241.</p> <p>Siebel Open UI uses this configuration for the icon. Use a comma to separate the abbreviation from the class name.</p> <p>You can configure more than one applet user property. For example, Pane 2 Icon 0, Pane 2 Icon 1, and so on.</p> |
| Pane 2 Item Icon Fields | DNMF,NF,DF,SF,FSF,SRF,HF,AF,2HHF,SFF | <p>Specify the abbreviations that you defined for the corresponding user properties in Step 3 on page 241. For example, specify the abbreviations for the Pane 2 Field 0 applet user property, the Pane 2 Field 1 applet user property, and so on. The abbreviations in this example come predefined with Siebel Hospitality. You cannot use any other abbreviation. You must use a different set of abbreviations for your Siebel application.</p> <p>Use a comma to separate each abbreviation.</p> |

- 6 Configure Drag and Drop.

Siebel Open UI uses a business service method to implement drag and drop functionality. This step describes how to specify the input arguments that this method requires. You add each of the following applet user properties.

| Name | Value | Description |
|--------------------------------|--|---|
| Disable Drag for Ganttchart | N | <p>Specify to allow the user to drag and drop items. Use one of the following values:</p> <ul style="list-style-type: none"> ■ Y. Allow drag and drop. ■ N. Do not allow drag and drop. |
| DragnDrop: Service Inputs 1 | "Service Name", "TNT Gantt UI Service" | Specify the business service that Siebel Open UI uses to handle a drag and drop operation. You must use this value. You cannot modify it. |

| Name | Value | Description |
|--------------------------------|---|--|
| DragnDrop: Service Inputs 2 | "Service Method", "DragnDrop" | Specify the business service method that Siebel Open UI uses to handle a drag and drop operation. You must use this value. You cannot modify it. |
| DragnDrop: Service Inputs 3 | "BO", "Quote" | Specify the business object. |
| DragnDrop: Service Inputs 4 | You can use these applet user properties to specify more input arguments that your deployment requires. | |
| DragnDrop: Service Inputs 5 | | |
| DragnDrop: Service Inputs 6 | | |
| DragnDrop: Service Inputs 7 | | |
| | | |

- 7 Configure other Schedule pane behavior, such as drilldown, extend, shrink, add, update, and delete. Add each of the following applet user properties, as required.

| Name | Value | Description |
|---|---|--|
| Create Task: Service Inputs 1 | "Service Name", "TNT Gantt UI Service" | Specify the business service that Siebel Open UI uses if the user clicks OK in the popup dialog box that it displays in the Schedule pane. |
| Create Task: Service Inputs 2 | "Service Method", "CreateBooking Record" | Specify the business service method that Siebel Open UI uses if the user clicks OK in a popup dialog box. |
| Disable Resize for Ganttchart | N | Specify to allow the user to resize an activity or a booking. Use one of the following values: ■ Y. Allow resizing. ■ N. Do not allow resizing. |
| ExtendShrink: Service Inputs 1 | "Service Name", "TNT Gantt UI Service" | Specify the business service that Siebel Open UI uses to handle a resize operation. |
| ExtendShrink: Service Inputs 2 | "Service Method", "ExtendShrink" | Specify the business service method that Siebel Open UI uses to handle a resize operation. |
| Pane 2 Disable ExtendShrink Views | : 32: 256: | Specify to disable resizing for a time scale. For example, 32 and 256 each represent a time_scale_identifier: ■ 32. Specifies the Month/Day-of-Week time scale. ■ 256. Specifies the Month/Day-of-Week/Day Part scale. Siebel Open UI uses a number to identify each time scale. For more information, see "Determining the Number That Siebel Open UI Uses to Identify Time Scales" on page 229. You must include a color before and after each identifier. |
| Show Task Details: Service Inputs 1 | "Service Name", "TNT Gantt UI Service" | Specify the business service that Siebel Open UI uses if the user double-clicks a booking, a task, or an activity, and then clicks OK in a popup dialog box. |
| Show Task Details: Service Inputs 2 | "Service Method", "CreateBooking Record" | Specify the business service method that Siebel Open UI uses if the user double-clicks a booking, a task, or an activity, and then clicks OK in a popup dialog box. |

| Name | Value | Description |
|----------------------------|--------------------|---|
| Pane 2 Item Drilldown Name | Activity Drilldown | <p>Specify the drilldown object that Siebel Open UI uses when the user clicks a label in the Schedule pane. Siebel Open UI navigates the user to the view that this drilldown object defines.</p> <p>This configuration works in conjunction with the DDID value that you configure in the Pane 2 Field <i>number</i> applet user property.</p> <p>You must configure the corresponding drilldown object in the applet.</p> |

Customizing Tooltips in Resource Schedulers

This topic describes how to customize the Tooltips that Siebel Open UI displays in a resource scheduler.

To customize tooltips in resource schedulers

- 1 In the Object List Editor, choose the applet that you modified in [Step 1 on page 218](#).
- 2 In the Object List Editor, expand the Applet tree, and then click Applet User Prop.
- 3 In the Applet User Props list, add each of the following applet user properties, as required.

| Name | Value | Description |
|------------------------------------|---------------------|---|
| Pane 2 Tooltip BC Name | TNT SHM FSI Booking | Specify the business component that Siebel Open UI uses to get the tooltip information for the events that it displays in the Schedule pane. This business component must contain the information that Siebel Open UI displays in the tooltip. |
| Pane 2 Tooltip BO Name | SHM Site | Specify the business object that references the business component that you specify in the Pane 2 Tooltip BC Name applet user property. |
| Pane 2 Tooltip Field <i>number</i> | Quote Name Tip | <p>Specify the business component fields that Siebel Open UI uses to get the information that it displays in the tooltips in the Schedule pane. Siebel Open UI adds a new line for each of these field values in the tooltips and displays them consecutively. For example:</p> <p>Event1 Hol i day resorts 10: 00 12: 00</p> |

| Name | Value | Description |
|------------------------------------|---------------------------------------|---|
| Pane 0 Tooltip BC Name | TNT Product - ISS Admin | Specify the business component that Siebel Open UI uses to get the tooltip information for the Resource pane. |
| Pane 0 Tooltip BO Name | SHM Site | Specify the business object that references the business component that you specify in the Pane 0 Tooltip BC Name applet user property. |
| Pane 0 Tooltip Field <i>number</i> | Physical Area Tip | Specify the business component field that Siebel Open UI uses to get the information that it displays in the tooltips for the Resource pane. |
| Pane 0 Tooltip Header Field | Name | Specify the business component field that Siebel Open UI uses to get the information that it displays in the first field in the tooltips for Resource pane. |
| Pane 1 Tooltip BC Name | TNT SHM Property Special Dates Action | Specify the business component that Siebel Open UI uses to get the information that it displays in the tooltips for the Timescale pane. |
| Pane 1 Tooltip BO Name | SHM Site | Specify the business object that references the business component that you specify in the Pane 1 Tooltip BC Name applet user property. |
| Pane 1 Tooltip Field <i>number</i> | Tooltip | Specify the business component field that Siebel Open UI uses to get the information that it displays in the tooltips for the Timescale pane. |
| Pane 1 Tooltip SortSpec | Type | Specify the sort specification that Siebel Open UI uses to sort the records in the business component that it uses to get the tooltip information for the Timescale pane. Siebel Open UI uses this configuration to sort sentences in a tooltip that includes more than one sentence. |
| EnableTooltip | Y | Specify to display or not display the tooltip. Use one of the following values: ■ Y. Display the tooltip. ■ N. Do not display the tooltip. |

- 4 Configure any special functionality that your tooltip deployment requires. Add each of the following applet user properties, as required.

| Name | Value | Description |
|-------------------------------|----------------------|--|
| Pane 2 Tooltip Service Method | GetEventTooltipInfo | Specify the business service method that Siebel Open UI uses to get the tooltip information for the Schedule pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties: <ul style="list-style-type: none"> ■ Pane 2 Tooltip BC Name ■ Pane 2 Tooltip BO Name ■ Pane 2 Tooltip Field <i>number</i> |
| Pane 2 Tooltip Service Name | TNT Gantt UI Service | Specify the business service name that Siebel Open UI uses to get the tooltip information for the Schedule pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties: <ul style="list-style-type: none"> ■ Pane 2 Tooltip BC Name ■ Pane 2 Tooltip BO Name ■ Pane 2 Tooltip Field <i>number</i> |
| Pane 1 Tooltip Service Method | GetTSTooltipInfo | Specify the business service method that Siebel Open UI uses to get the tooltip information for the Timescale pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties: <ul style="list-style-type: none"> ■ Pane 1 Tooltip BC Name ■ Pane 1 Tooltip BO Name ■ Pane 1 Tooltip Field <i>number</i> |
| Pane 1 Tooltip Service Name | TNT Gantt UI Service | Specify the business service that Siebel Open UI uses to get the tooltip information for the Timescale pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties: <ul style="list-style-type: none"> ■ Pane 1 Tooltip BC Name ■ Pane 1 Tooltip BO Name ■ Pane 1 Tooltip Field <i>number</i> |

| Name | Value | Description |
|-------------------------------|----------------------|--|
| Pane 0 Tooltip Service Method | GetResTooltipInfo | <p>Specify the business service method that Siebel Open UI uses to get the tooltip information for the Resource pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties:</p> <ul style="list-style-type: none"> ■ Pane 0 Tooltip BC Name ■ Pane 0 Tooltip BO Name ■ Pane 0 Tooltip Field <i>number</i> |
| Pane 0 Tooltip Service Name | TNT Gantt UI Service | <p>Specify the business service that Siebel Open UI uses to get the tooltip information for the Resource pane. If you do not specify this applet user property, then Siebel Open UI calls the default business service, and then displays data according to the configurations of the following user properties:</p> <ul style="list-style-type: none"> ■ Pane 0 Tooltip BC Name ■ Pane 0 Tooltip BO Name ■ Pane 0 Tooltip Field <i>number</i> |

8

Configuring Siebel Open UI to Interact with Other Applications

This chapter describes how to configure Siebel Open UI to interact with other applications. It includes the following topics:

- [Displaying Data from External Applications in Siebel Open UI on page 253](#)
- [Displaying Data from Siebel Open UI in External Applications on page 260](#)

Displaying Data from External Applications in Siebel Open UI

This topic describes how to configure Siebel Open UI to interact with other applications. It includes the following information:

- [Displaying Data from External Applications in Siebel Views on page 253](#)
- [Displaying Data from External Applications in Siebel Applets on page 256](#)

Displaying Data from External Applications in Siebel Views

The example in this topic describes how to configure Siebel Open UI to get connection details from LinkedIn, find matching mutual contacts in Affiliation views, and then display the matching records in a Siebel view.

To display data from external applications in Siebel views

- 1 Set up the data:
 - a Log in to LinkedIn, and then identify two connections that include profile pictures and that allow you to reference them in your configuration.
 - b Write down the case-sensitive first name and last name for each LinkedIn profile.
 - c Log in to Siebel Call Center, navigate to the contacts Screen, and then the Contact List view.
 - d Click New, and then enter the First Name and Last Name values for one of the profiles that you noted in [Step b](#).

The values you enter must match exactly. Make sure uppercase and lowercase usage is the same.
 - e Click New, and then enter the First Name and Last Name values for the other profile you noted in [Step b](#).
 - f Navigate to the Opportunity screen, and then the Opportunity List view.

- g** Click New to create a new opportunity, and then add the contact that you created in [Step d](#) to this new opportunity.
- h** Click New to create another new opportunity, and then add the contact that you created in [Step e](#) to this new opportunity.
- i** Log in to the Siebel application using the sample database, and then repeat [Step b](#) through [Step e](#).
- j** Navigate to the Contact Screen, and then the Contact List view.
- k** Drill down on the first contact, and then navigate to the third level Affiliations view.
- l** Click New, and then add the contact that you created in [Step d](#).
- m** Click New, and then add the contact that you created in [Step e](#).

- 2** Replace the SRF that the Mobile Web Client uses in the following folder:

`CLIENT_HOME\Objects\enu`

This SRF includes the Siebel Tools modifications that this example requires.

- 3** Download the sociallyawaremodel.js file into the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\Language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. This code already contains the configuration that Siebel Open UI requires to authenticate the user with LinkedIn and to get the connections for this user from LinkedIn.

- 4** Use a JavaScript editor to open the sociallyawaremodel.js file that you downloaded in [Step 3](#).
- 5** Locate the following code:

```
SociallyAwarePM.prototype.Init = function(){  
    SiebelAppFacade.SociallyAwarePM.superclass.Init.call(this);
```

- 6** Add the following code immediately under the code you located in [Step 5](#):

```
this.AddProperty("LinkedInRecordSet", []);  
this.AddProperty("LinkedInMarker", 0);
```

where:

- `LinkedInRecordSet` stores the connection details of the current user from LinkedIn.
- `LinkedInMarker` marks the position in the connection details record set for querying purposes in the Siebel Database.

- 7** Add the following code immediately after the code you added in [Step 6](#):

```
this.AddMethod("QueryForRelatedContacts", QueryForRelatedContacts);  
this.AddMethod("GetConnectionByName", GetConnectionByName);
```

This code allows the presentation model to call the `GetConnectionByName` method and the `QueryForRelatedContacts` method that you add in [Step 8](#).

- 8** Add the following code immediately after the `FetchConnectionFromLinkedIn` method:

```
function GetConnectionByName(fName, lName){
    var connection = null;
    if(fName && lName){
        var linkedInRecSet = this.Get("linkedInRecordSet");
        for(var i = 0; i < linkedInRecSet.length; i++){
            var current = linkedInRecSet[i];
            if(current.firstName === fName && current.lastName === lName)
                {connection = current; break; }
        }
    }
    return connection;
}

function QueryForRelatedContacts(){
    var currentMark = this.Get("linkedInMarker");
    var recordSet = this.Get("linkedInRecordSet");
    var firstName = "";
    var lastName = "";
    for(var i = currentMark; i < currentMark + 5; i++){
        var current = recordSet[i];
        firstName = firstName + current["firstName"];
        lastName = lastName + current["lastName"];
        if(i < (currentMark + 4))
            {firstName = firstName + " OR ";
            lastName = lastName + " OR ";
        }
    }
    if(firstName !== "" || lastName !== ""){
        SiebelApp.S_App.GetActiveView().ExecuteFrame(
            this.Get("GetName"),
            [
                {field : "Last Name", value : lastName},
                {field : "First Name", value : firstName}]);
    }
}
```

where:

- GetConnectionByName uses the first name and last name to get the connection information stored on the client. Siebel Open UI gets this information from LinkedIn.
- QueryForRelatedContacts is the presentation model method that uses the subset of the LinkedIn connection record that Siebel Open UI sets to query the Siebel Server for matching records. The notification causes Siebel Open UI to call the BindData method of the physical renderer as part of the reply processing. The BindData method updates the user interface with the matching set of records from server. For more information, see [“Notifications That Siebel Open UI Supports” on page 541](#).

- 9 Add the following code immediately below the AddProperty methods you added in [Step 6](#):

```
this.AddMethod("QueryForRelatedContacts", QueryForRelatedContacts);
this.AddMethod("GetConnectionByName", GetConnectionByName);
```

These AddMethod calls add the QueryForRelatedContacts method and the GetConnectionByName method so that Siebel Open UI can call them from the presentation model.

- 10 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.
- For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- b** Navigate to the Administration - Application screen, and then the Manifest Files view.

- c** In the Files list, add the following file.

| Field | Value |
|-------|--------------------------------------|
| Name | siebel/custom/sociallyawarepmodel.js |

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.

- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|--------------------|
| Type | Applet |
| Usage Type | Presentation Model |
| Name | Enter any value. |

- f** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 1 |

- g** In the Files list, add the following file:

siebel/custom/sociallyawarepmodel.js

- 11** Test your modifications.

Displaying Data from External Applications in Siebel Applets

The example in this topic describes how to configure Siebel Open UI to display data from an external application in a Siebel applet. Siebel Open UI can use a symbolic URL open this external application from an applet. For example, to display a Google Map or a Linked In view as an applet in a Siebel application.

The example in this topic configure Siebel Open UI to display a Google map as a child applet in the Account detail page. The Map displays a location according to the Zip Code of the account record. If the Zip Code is empty, then it displays the default Google map.

To display data from external applications in Siebel applets

1 Configure the business component:

- a** Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b** In the Object Explorer, click Business Component.

- c** In the Business Components list, query the Name property for Account.

- d** In the Object Explorer, expand the Business Component tree, and then click Field.

- e** In the Fields list, add the following field.

| Property | Value |
|------------------|---|
| Name | You can use any value. For this example, use the following value: Symbol i cURLGoogl eMap |
| Calculated | TRUE |
| Type | DTYPE_TEXT |
| Calculated Value | Enter the name of any Symbolic URL enclosed in double quotation marks. For this example, enter the following value: Symbol i cURLGoogl eMap You define this Symbolic URL later in this example. |

2 Configure the applet:

- a** In the Object Explorer, click Applet.

- b** In the Applets list, query the Name property for SSO Analytics Administration Applet.

In a typical configurator, you create an applet that Siebel Open UI can use to display the external content. This applet must reference the business component that you configured in [Step 1](#).

- c** Copy the applet that you located in [Step b](#), and then set the following properties for this copy.

| Property | Value |
|--------------------|-----------|
| Name | GoogleMap |
| Business Component | Account |
| Title | GoogleMap |

- d** In the Object Explorer, expand the Applet tree, expand the List tree, and then click List Column.

- e In the List Columns list, set the following properties for the single record that the list displays.

| Property | Value |
|----------------------|----------------------|
| Name | SymbolicURLGoogleMap |
| Field | SymbolicURLGoogleMap |
| Field Retrieval Type | Symbolic URL |

3 Configure the view:

- a In the Object Explorer, click View.
- b In the Views list, query the Name property for the view that must display the Google map.

For this example, query the Name property for the following value:

Account Detail - Contacts View

- c In the Object Explorer, expand the View tree, expand the View Web Template tree, and then click View Web Template Item.
- d In the View Web Template Items list, add the following view web template item.

| Property | Value |
|----------------------|--|
| Name | GoogleMap |
| Applet | GoogleMap |
| Field Retrieval Type | Symbolic URL |
| Item Identifier | Enter the next highest number in the sequence of numbers that Siebel Tools displays for all records in the View Web Template Items list. |

Note that you cannot drag, and then drop an applet into the Web Layout Editor in Siebel Tools. You must add it manually to the web page.

- 4** Compile your modifications.
- 5** Examine the URL that Siebel Open UI must integrate:

- a Open the URL that Siebel Open UI must integrate.
- For this example, open <http://maps.google.com/> in a browser.

- b View the source HTML.
- For example, if you use Internet Explorer, then click the View menu, and then click Source. Alternatively, save the file to your computer, and then use an HTML editor to open it.

- c Identify the input fields.

It is recommended that you search for the input tag. In this example, the source displays the name in the following way:

name="q"

You use this value when you define the arguments for the Symbolic URL.

- d Determine if the method attribute of the page is one of the following:
 - **POST.** You must define the PostRequest command as an argument of the symbolic URL.
 - **GET.** you do not need to define a symbolic URL command.

In this example, the method is GET.

- e Determine the target of the from action attribute, which is typically specified as action = "*some string*". In this situation, it is '/maps'. It is appended to the predefined URL.

6 Configure the symbolic URL:

- a Log in to the Siebel client with administrator privileges.
- b Navigate to the Administration - Integration screen, and then the WI Symbolic URL List view.
- c In the Fixup Administration dropdown list, choose Symbolic URL Administration.
- d In the Symbolic URL Administration list, add the following symbolic URL.

| Field | Value |
|-----------------|-----------------------------|
| Name | SymbolicURLGoogleMap |
| URL | http://maps.google.com/maps |
| Fixup Name | Default |
| SSO Disposition | IFrame |

- e In the Symbolic URL Arguments list, add the following symbolic URL argument.

| Field | Value |
|-------------------|---|
| Name | q This value is the input tag in HTML for the Google map. |
| Required Argument | N You set this argument to N because the account might not include a zip code. |
| Argument Type | Field Siebel Open UI must send the value in the zip code field of the account to the Google map. |
| Argument Value | Postal Code You set this argument to the name of the business component field that contains the value that Siebel Open UI must send to the Google map. |

| Field | Value |
|--------------------|-------|
| Append as Argument | Y |
| Substitute in Text | N |
| Sequence# | 1 |

- f** In the Symbolic URL Arguments list, add the following symbolic URL arguments. Siebel Open UI uses this argument to embed the Google map in the applet.

| Field | Value |
|--------------------|----------|
| Name | output |
| Required Argument | Y |
| Argument Type | Constant |
| Argument Value | embed |
| Append as Argument | Y |
| Substitute in Text | N |
| Sequence# | 2 |

- 7** Test your modifications:
- a** Navigate to the Accounts screen, and then click Accounts List.
 - b** In the Accounts List, create a new account and include a value in the Zip Code field.
 - c** Drill down on the Account Name field.
 - d** Make sure Siebel Open UI displays a Google map and that this map includes a push pin that identifies the zip code that you entered in [Step b](#).

Displaying Data from Siebel Open UI in External Applications

This topic describes how to display data from Siebel Open UI in an external application. It includes the following information:

- [“Displaying Siebel Portlets In External Applications” on page 261](#)
- [“Preparing Standalone Applets” on page 266](#)
- [“Displaying Siebel CRM Applets and Views in External Applications” on page 267](#)
- [“Displaying Siebel CRM Applets in External Applications with Search Criteria” on page 269](#)
- [“Using iFrame Gadgets to Display Siebel CRM Applets in External Applications” on page 272](#)
- [“Displaying Siebel CRM Applets in Siebel CRM Web Pages” on page 271](#)

■ [Using iFrame Gadgets to Display Siebel CRM Applets in External Applications on page 272](#)

Siebel Open UI comes predefined to display Siebel CRM data only in a Siebel application, such as Siebel Call Center. This topic describes how to display Siebel CRM data in an external application or website, such as Oracle WebCenter or iGoogle.

Displaying Siebel Portlets In External Applications

This topic describes how to display a Siebel portlet in an external application. It includes the following information:

- ["Specifying GotoView or GetApplet" on page 263](#)
- ["Specifying URLs to Siebel Portlets" on page 264](#)
- ["Using Cascading Style Sheets Instead of iFrame Attributes" on page 266](#)

You can configure Siebel Open UI to display a Siebel portlet. A *Siebel portlet* is a Siebel application that Siebel Open UI embeds in a thirty-party website. Oracle WebCenter and iGoogle are examples of third-party websites. Siebel Open UI uses an HTML iFrame to display a Siebel portlet in a Siebel portlet window. You can configure Siebel Open UI to make more than one call to Siebel portlets only when it calls the GotoView method or the GetApplet method. For more information about the different methods that you can use in a Siebel portlet, see *Siebel Portal Framework Guide*.

To display Siebel portlets in external applications

- 1 Set the server parameters:
 - a Log in to the Siebel client with administrative privileges.
 - b Navigate to the Administration - Server Configuration screen, and then the Servers view.
 - c In the Siebel Servers list, choose a Siebel Server.
 - d Click Parameters.
 - e In the Parameters list, add the following parameters.

| Parameter | Description |
|-------------------|--|
| PortletAPIKey | Enter any value. For more information about the values that you can use when configuring authentication, see <i>Siebel Security Guide</i> . |
| PortletOriginList | Specify the list of domains that can send a communication request to the Siebel portlet. To specify more than one domain, you can use a comma to separate each domain. If the Siebel portlet receives a communication request from a domain that PortletOriginList does not specify, then Siebel Open UI creates an error in the Siebel portlet. |

| Parameter | Description |
|---------------------------|--|
| PortletMaxAllowedAttempts | <p>Specify the number of unsuccessful times that the portlet can call a method before Siebel Open UI blocks any subsequent calls. An unsuccessful call can occur in the following situations:</p> <ul style="list-style-type: none">■ A domain attempts to send a communication request to the portlet, but the PortletOriginList does not specify this domain.■ The portlet_key is not valid. <p>Siebel Open UI resets the unsuccessful attempts to zero after the time interval that the PortletBlockedInterval server parameter expires. Siebel Open UI comes predefined with the PortletMaxAllowedAttempts server parameter set to 3.</p> |
| PortletBlockedInterval | <p>Specify how long the portal remains blocked. Siebel Open UI comes predefined with the PortletBlockedInterval server parameter set to 900.</p> |

- 2 Add the following object to your custom code. The SWEView, SWEApplet, and Key arguments are required. All other arguments are optional:

```
var msg = new Object();  
msg.SWEView = view_name;  
msg.SWEApplet = applet_name;  
msg.SWECmd = GotoView or GetApplet  
msg.Key = portlet_key;
```

where:

- *view_name* specifies the view that Siebel Open UI displays in the portlet window. If you specify only the view, then Siebel Open UI displays the view and all the applets that this view contains.
- *applet_name* specifies the applet that Siebel Open UI displays in the portlet window. If you specify only the applet, then Siebel Open UI displays only this applet and no view. If you specify the view and applet, then Siebel Open UI displays the applet in the view.
- *GotoView or GetApplet* specifies whether or not to display a view or an applet in the portlet window. For more information, see ["Specifying GotoView or GetApplet" on page 263](#).
- *portlet_key* must specify the value that you specify for the PortletAPIKey server parameter in [Step 1](#). The Siebel client sends this value to the Siebel Server when it calls a Siebel application. You must include the msg.Key argument, and the value of this argument must match the value of the key that the PortletAPIKey server parameter contains on the Siebel Server. If the messaging object does not contain a key, or if it contains a key that does not match the value of the server parameter, then Siebel Open UI displays an error in the Siebel portlet.

For example, the following code displays the Opportunity List Applet, and it displays this applet inside the Opportunity List View:

```
var msg = new Object();
msg.SWEView = Opportunity List View;
msg.SWEApplet = Opportunity List Applet;
msg.Key = oracle123;
```

- 3** Add the following code immediately after the code that you added in [Step 2](#).

```
document.getElementById('siebel frameid').contentWindow.postMessage(msg, '*');
```

This code calls the method that the Siebel portlet uses.

- 4** Add the iFrame that calls the Siebel portlet.

You add this iFrame to a web page that resides in the third-party application. For example, the following code uses an iFrame to embed the Quote List Applet of Siebel Call Center in an external website:

```
<html>
<head>
<script>
</script>
</head>
<body>
<b><text>
<h2>Portlet public API demo</h2></text><b>
<IFRAME id="i19" name= "i19" src= "http://server_name.example.com/
callcenter_enu/
start.swe?SWECmd=ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password&SWE
AC=SWECmd=GetApplet&SWEApplet=Quote+List+Applet&ISPortlet=1&SWESM=Edit+List"
style="height: 50%; width: 100%; &KeepAlive=1&PtId=my_theme">
</IFRAME>
<table style="width: 100%; height: 50%">
<tr style="width: 100%; height: 100%">
<td style="width: 50%; height: 100%">
<IFRAME id="d121" name= "i19111" src= "http://server_name.example.com/
callcenter_enu/API Script.html "
style="height: 100%; width: 100%; ">
</IFRAME>
</td>
</tr>
</table>
</body>
</html>
```

Siebel Open UI can now create a Siebel session for this instance of the Siebel application in the browser. The browser can send more than one request to the same Siebel session without refreshing the page. For more information about this iFrame, see [“Specifying URLs to Siebel Portlets” on page 264](#).

Specifying GotoView or GetApplet

As an option, you can specify GotoView or GetApplet.

To specify GotoView

- You must specify msg.SWEView.
- You can specify msg.SWEApplet as an option. If you include msg.SWEApplet, then Siebel Open UI displays only the applet that you specify.

For example, the following code displays only the Opportunity List Applet:

```
var msg = new Object();
msg.SWEView = Opportunity List View;
msg.SWEApplet = Opportunity List Applet;
msg.SWECmd=GotoView
msg.Key = oracle123;
```

For example, the following code displays the entire Opportunity List View:

```
var msg = new Object();
msg.SWEView = Opportunity List View;
msg.SWECmd=GotoView
msg.Key = oracle123;
```

To specify Getapplet

- You must specify msg.SWEApplet.
- You must configure the applet that msg.SWEApplet specifies as a standalone applet. For more information, see [“Preparing Standalone Applets” on page 266](#).
- msg.SWEView is not required. Do not specify it.

For example, the following code displays only the Opportunity List Applet:

```
var msg = new Object();
msg.SWEApplet = Opportunity List Applet;
msg.SWECmd=GetApplet
msg.Key = oracle123;
```

Specifying URLs to Siebel Portlets

You can use the following code to specify the Siebel URL that the external application uses to access the Siebel Server:

```
<HTML>
  <BODY>
    <IFRAME src = "http://server_address/application/
start.swe?SWECmd=SWECmd=Start&isPortlet=1&other_arguments"> </IFRAME>
  </BODY>
</HTML>
```

where:

- *server_address* specifies the address of the Siebel Server.
- *application* specifies the Siebel application.
- SWECmd is a required argument that specifies how to display the Siebel application when the user accesses this URL.

- `isPortlet` is a required argument that informs the Siebel Server that this application runs in a portlet. The server requires this argument so that it can do the processing it requires to support a portlet.
- `other_arguments` specify how to display the Siebel application. For example, the login requirements to display, the applets to display, how to size applets, and so on.

For example, consider the following iFRAME src:

```
http://server_name.example.com/callcenter_enu/
start.swe?SWECmd=ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password&SWE
AC=SWECmd=GetApplet&SWEApplet=Quote+List+Applet&IsPortlet=1&SWESM=Edit+List"style="sty
le="height: 50%;width: 100%; &KeepAlive=1&PtlD=my_theme"
```

Table 16 describes the parts of this iFRAME src that specifies the Siebel URL.

Table 16. Specifying URLs to Siebel Portlets

| URL Argument | Description |
|---|---|
| <code>http://server_name.example.com</code> | Access the Siebel Server that resides at <code>server_name.example.com</code> . |
| <code>/callcenter_enu</code> | Run the CallCenter application. |
| <code>/start.swe?</code> | Start the Siebel Web Engine. |
| <code>SWECmd=</code> | Provide commands to the Siebel Web Engine. |
| <code>ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password</code> | Provide the user name and password authentication arguments. |
| <code>&SWEAC=SWECmd=GetApplet&SWEApplet=Quote+List+Applet</code> | SWEAC specifies more than one command. This example specifies to display the Quote List Applet. |
| <code>&IsPortlet=1</code> | Run the CallCenter application as a portlet. |
| <code>&SWESM=Edit+List"style="height: 50%;width: 100%;</code> | Use the Edit List style, and set this style to 50% of the predefined height and 100% of the predefined width. |

Table 16. Specifying URLs to Siebel Portlets

| URL Argument | Description |
|------------------|---|
| &KeepAlive=1 | Keep sessions active even if the session is not in use. Siebel CRM comes predefined to expire a Siebel session that is not in use for a period of time according to the value that the SessionTimeout server parameter specifies. If this session times out, and if you do not specify KeepAlive=1, then Siebel Open UI displays a login dialog box in the client. This behavior might not be desirable in a Siebel portlet. It is recommended that you set this argument to keep the session active. |
| &PtlId=my_theme" | <p>Use cascading style sheets instead of iFrame attributes. For more information, see "Using Cascading Style Sheets Instead of iFrame Attributes" on page 266.</p> <p>You must define a theme in the theme.js file and then use the PtlId argument to reference it. In this example, my_theme is the name of the theme that the theme.js file contains.</p> |

Using Cascading Style Sheets Instead of iFrame Attributes

This iFrame tag supports a number of attributes that you can use to control how Siebel Open UI displays the portlet content. For more information about these attributes, see the page that describes HTML element usage in an iFrame at the following W3C website:

<http://www.w3.org/wiki/HTML/Elements/iframe>

Recent HTML versions might replace these attributes. So, it is strongly recommended that you use cascading style sheets instead of these attributes. Siebel Open UI comes predefined to use cascading style sheet classes for this iFrame. For more information, see ["Using Cascading Style Sheet Classes to Modify HTML Elements" on page 153](#).

Preparing Standalone Applets

A *standalone applet* is a type of applet that Siebel Open UI can display outside the context of a Siebel CRM view. A predefined view references a business object, a business object references a business component, and an applet also references a business component, but an applet does not reference a business object in a predefined Siebel Open UI configuration. You must modify this configuration so that the applet can work independently of the view. To do this, you configure the applet to directly reference the business object.

To prepare standalone applets

- 1 Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- 2 In the Object Explorer, click Applet.

- 3 In the Applets list, query the Name property for the applet that Siebel Open UI must display outside of the view.
- 4 In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
- 5 In the Applet User Properties list, add the following applet user property.

| Property | Description |
|----------|--|
| Name | Enter the following value: Business Object |
| Value | Enter the name of the business object that this applet must reference. |

Displaying Siebel CRM Applets and Views in External Applications

This topic describes how to display a Siebel CRM applet or view in an external application, such as Oracle WebCenter, iGoogle, and so on.

To display Siebel CRM applets and views in external applications

- 1 Prepare the applet.

For more information, see ["Preparing Standalone Applets" on page 266](#).

- 2 Determine where to add the Siebel URL in the external application.

To display a Siebel CRM applet or view in an external application, you add an iFrame that includes the Siebel URL. You add this iFrame to the external application. For example, you can add the following iFrame to the main Web page of the external application:

```
<HTML>
  <BODY>
    <I FRAME src = "Siebel URL"> </I FRAME>
    <I FRAME src = "http://www.example.com"> </I FRAME>
  </BODY>
</HTML>
```

This example uses an iFrame to expose the following portlets:

- One portlet runs the Siebel application.
 - One portlet specifies the Oracle public website.
- 3 Add the following Siebel URL to the external application that you identified in [Step 2](#):

```
http://server_name/virtual_folder/
start.swe?SWECmd=ExecuteLogi n&SWEUserName=user_name&SWEPassword=my_password&SWE
AC=SWECmd=GetAppl et&SWEAppl et=appl et_name&ISPortl et=1&SWESM=appl et_mode
```

where:

- *user_name* specifies the user name that identifies the person who uses the Siebel CRM client.

- *my_password* specifies the password.
- *applet_name* specifies the name of the applet that Siebel CRM must display in the external application.
- *applet_mode* specifies the mode that Siebel Open UI uses to display the applet. You can use one of the following values:
 - Edit+List
 - Edit
 - Base
 - New
 - Query

For more information about these modes, see [“Displaying Applets Differently According to the Applet Mode” on page 190](#).

For example, the following URL configures Siebel Open UI to log in to the Siebel application as the TEST user, and then to display the Contact List Applet in Edit mode:

```
<HTML>
  <BODY>
    <IFRAME src = "http://server_name.example.com/calcenter_enu/
start.swe?SWECmd=ExecuteLogi n&SWEUserName=TEST&SWEPassword=my_password&SWEAC=SW
ECmd=GetAppl et&SWEAppl et=Contact+Li st+Appl et&ISPortl et=1&SWESM=Edi t"> </IFRAME>
    <IFRAME src = "http://www.example.com"> </IFRAME>
  </BODY>
</HTML>
```

You must write each argument so that it complies with URL standards. For example:

- If the applet name includes spaces, then you must replace each space with the plus (+) symbol.
- If the display mode includes a space, then you must replace each space with the plus (+) symbol. For example, Edit+List.

If you do not include the *user_name* and *my_password*, then Siebel Open UI displays the login page. If the user logs in successfully, then Siebel Open UI displays the applet.

For other ways to configure the URL, see [“Alternatives to Configuring the URL” on page 269](#).

The work you must do to add a URL to an external application varies depending on each application. For an example, see [“Using iFrame Gadgets to Display Siebel CRM Applets in External Applications” on page 272](#).

Alternatives to Configuring the URL

Table 17 describes alternative URL configurations.

Table 17. Alternatives to Configuring the URL

| URL | Description |
|--|--|
| <code>http://server_name/virtual_folder/start.swe?SWECmd=ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password&SWEAC=SWECmd=GotoView&SWEView=view_name</code> | This URL displays a view. |
| <code>http://server_name/virtual_folder/start.swe?SWECmd=ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password&SWEAC=SWECmd=GotoView&SWEView=view_name&SWEApplet=applet_name</code> | This URL displays a predefined applet. |
| <code>http://server_name/virtual_folder/start.swe?SWECmd=ExecuteLogin&SWEUserName=user_name&SWEPassword=my_password&SWEAC=SWECmd=GetApplet&SWEApplet=applet_name</code> | This URL displays a standalone applet. For more information, see “Preparing Standalone Applets” on page 266. |

Alternative to Using SweCmd=ExecuteLogin

You can use SweCmd=GetApplet or SweCmd=GotoView as an alternative to using SweCmd=ExecuteLogin. For example:

```
http://server_name.example.com/callcenter_enu/
start.swe?SweCmd=GetApplet&SWEApplet=Contact+List+Applet&ISPortlet=1&SWESM=Edit
```

This alternative does not use the SWEAC argument. It configures Siebel Open UI to display the login page and then the applet. Note the following:

- GotoView can display an entire view or only an applet. It does not require Siebel Open UI to prepare the applet separately.
- GetApplet displays only an applet.

Displaying Siebel CRM Applets in External Applications with Search Criteria

This topic describes how to configure Siebel Open UI to display Siebel CRM applets in external applications with a search criteria so that the applet only displays records that meet this criteria.

To display Siebel CRM applets in external applications with search criteria

- 1 Prepare the applet that Siebel Open UI must display outside of the view.
For more information, see [“Preparing Standalone Applets”](#) on page 266.
- 2 Add the following URL to the external application:

```
http://server_name/virtual_folder/  
start.swe?SWECmd=ExecuteLogi n&SWEUserName=user_name&SWEPassword=my_password&SWE  
AC=SWECmd=GetAppl et&SWEAppl et=appl et_name&I sPortl et=1&SWESM=di spl ay_mode&BCFi el  
d0=fi el d_name&BCFi el dVal ue0=fi el d_val ue&BCFi el d1=fi el d_name&BCFi el dVal ue1=fi el d  
_val ue&PBCFi el d0=parent_fi el d&PBCFi el dVal ue0=parent_fi el d_val ue
```

where:

- *field_name* identifies the name of the business component field that Siebel Open UI searches.
- *field_value* specifies the value that Siebel Open UI uses to do the search.
- *parent_field* is an optional argument that identifies the name of a field that resides in the parent business component that Siebel Open UI searches. For more information about parent business components, see *Configuring Siebel Business Applications*.
- *parent_field_value* is an optional argument that specifies the value that Siebel Open UI uses to do the search in the parent business component.

For example, the following URL configures Siebel Open UI to log in to the Siebel application as the TEST user, to display the Contact List Applet in Edit mode, and to only display records in this applet that include a value of Oracle in the Account field:

```
http://server_name.example.com/callcenter_enu/  
start.swe?SWECmd=ExecuteLogi n&SWEUserName=TEST&SWEPassword=my_password&SWEAC=SW  
ECmd=GetAppl et&SWEAppl et=Contact+Li st+Appl et&I sPortl et=1&SWESM=Edi t&BCFi el d0=Ac  
count&BCFi el dVal ue0=0racl e
```

Note the following:

- A field value can contain any data that the Search Specification property of the business component in Siebel Tools specifies. For example, the following search specification configures Siebel Open UI to display all records that include a value of Account1 or Account2 in the BCField0 field:

PBCFi el d0=Account1+OR+Account2
- The URL can include a field value that Siebel Open UI does not display in the applet. The search that Siebel Open UI performs can activate and use these fields.
- Siebel Open UI uses the search specification to determine the records it displays according to the view navigation that the user performs.
- If you include a parent business component field in the URL, and if the business component where this field resides does not have a parent business component, then Siebel Open UI ignores this field in the URL.
- If you configure a parent business component field in the URL, and if this field does not exist in the business component that you specify, then the URL is not valid, Siebel Open UI will not build the applet, and unpredictable results might occur in the portlet.
- The work you must do to add a URL to an external application varies depending on each application. For an example, see [“Using iFrame Gadgets to Display Siebel CRM Applets in External Applications” on page 272](#).

Displaying Siebel CRM Applets in Siebel CRM Web Pages

This topic describes how to display a Siebel CRM applet in the web page of a Siebel application.

To display Siebel CRM applets in Siebel CRM web pages

- 1 Prepare the standalone applet.

For more information, see [“Preparing Standalone Applets” on page 266](#).

- 2 Open Windows Explorer on the Siebel Server, and then navigate to the following folder:

siebsrvr/webtempl /oui webtempl

- 3 Use an XML editor to open the web page that you must modify.

For example, open the CCFrameContentHI.swt file that resides in the ouiwebtempl folder. This template determines the content that Siebel Open UI displays. You can modify any template that resides at the page level. You typically modify the CCPageContainer.swt file page container template.

- 4 Add the following code, and then save your modifications:

```
<swe: applet id="10000" hintText="Stand alone applet" var="Parent">
  <swe: this property="FormattedHtml " />
</swe: applet>
```

You can add this code anywhere in the application template. You typically position it beside the _sweview swe:frame in the CCFrameContentHI.swt file.

- 5 Modify the web page:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b In the Object Explorer, click Web Page.

- c In the Web Page list, query the Name column for the web page that you modified in [Step 3](#).

- d In the Object Explorer, expand the Web Page tree, and then click Web Page Item.

- e In the Web Page Items list, add the following web page item.

| Property | Value |
|-----------------|---|
| Name | Enter the name of the applet that Siebel Open UI must display in the web page. |
| Type | Applet |
| Item Identifier | Enter the number that the Id attribute that resides in the template file specifies. For example, enter the following value: 1000 |

- f In the Object Explorer, expand the Web Page Item tree, and then click Web Page Item Parameter.

- g** In the Web Page Item Parameter list, add the following web page item parameter.

| Property | Value |
|----------|--|
| Name | Mode |
| Value | Enter one of the following values: <ul style="list-style-type: none">■ Edit List■ Edit■ Base■ New■ Query |

- h** Compile your modifications.

Using iFrame Gadgets to Display Siebel CRM Applets in External Applications

The example in this topic describes how to use iFrame gadgets to configure Siebel Open UI to display a Siebel applet in an external application.

To use iFrame gadgets to display Siebel CRM applets in external applications

- 1 Do the setup:
 - a** Create a LinkedIn profile at the <http://www.linkedin.com> Web site.
 - b** Create a Gmail profile at the <http://www.google.com/ig> Web site.
- 2 Configure the external applications:
 - a** Open a new browser session, navigate to <http://www.linkedin.com/>, and then log in to your profile:
 - b** Open a new browser tab, navigate to <http://www.google.com/ig>, and then log in to your gmail profile:
 - c** Navigate to <http://www.google.com/ig/settings>.
 - d** Click Add More Gadgets.
 - e** In the Search for Gadgets section, enter iFrame Gadget, and then click Search.
 - f** In the Search Results for the iFrame Gadget list, click iFrame Gadget.
 - g** Click Embed This Gadget.
 - h** In the Add This Gadget to Your Webpage page, enter the following URL that Siebel Open UI uses to display the applet. You enter this URL into the Address of Page to Show field:


```
http://server_name/callcenter_enu/  
start.swe?SWEUserName=user_name&SWEPassword=user_password&SWECmd=ExecuteLogi  
n&SWEAC=SWECmd=GotoView&SWEView=view_name&ISPortlet=1&SWEApplet=applet_name
```

where:

- *server_name* identifies the name of the server.
- *user_name* identifies the user name.
- *user_password* identifies the user password.
- *view_name* identifies the name of the view that contains the applet.
- *applet_name* identifies the applet that Siebel Open UI must display in the external application.

For example, you enter the following URL to display the Opportunity list applet:

```
http://server_name.example.com/callcenter_enu/  
start.swe?SWEUserName=%48%4B%49%4D&SWEPassword=%48%4B%49%4D&SWECmd=ExecuteLo  
gin&SWEAC=SWECmd=GotoView&SWEView=Opportunity+List+View&ISPortlet=1&SWEApple  
t=Opportunity+List+Applet
```

This URL configures the gadget to load the Opportunity applet from the server that this URL specifies. It uses an encrypted user name and password, represented as the following:

%48%4B%49%4D

It is strongly recommended that you use Web Single Sign-On (SSO) to handle this user name and password authentication. For more information, see the topic about the URL Login in *Siebel Security Guide*.

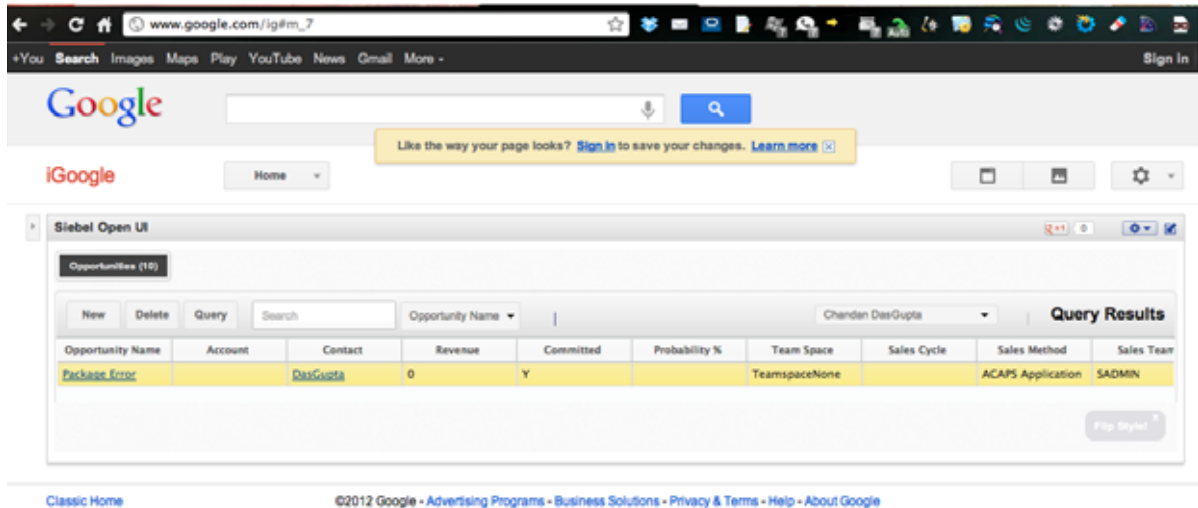
i Click Preview Changes.

j Click Save.

3 Test your modifications:

- a** Verify that iGoogle refreshes the page and displays the Opportunity list.
- b** Expand the widget to full screen to display the full width of the list.
- c** To choose a LinkedIn contact, use the menu that Google displays on the list header on the right side of the screen.
- d** Verify that the Web browser displays the opportunities for the contact that you choose.

- e Verify that the chosen LinkedIn contact matches a Siebel contact record.
Make sure the Web browser displays a layout that is similar to the following layout.



The screenshot shows a web browser window with the address bar set to `www.google.com/ig#m_7`. The page displays the Google iGoogle interface. A Siebel Open UI widget is embedded in the page, showing a table of query results for 'Opportunity Name'. The table has columns: Opportunity Name, Account, Contact, Revenue, Committed, Probability %, Team Space, Sales Cycle, Sales Method, and Sales Team. The first row of data shows 'Package Error' for Opportunity Name, 'DasGupta' for Contact, '0' for Revenue, 'Y' for Committed, and 'TeamSpaceNone' for Team Space. The table is titled 'Query Results' and includes a 'Flip Styler' button at the bottom right.

| Opportunity Name | Account | Contact | Revenue | Committed | Probability % | Team Space | Sales Cycle | Sales Method | Sales Team |
|------------------|---------|----------|---------|-----------|---------------|---------------|-------------|-------------------|------------|
| Package Error | | DasGupta | 0 | Y | | TeamSpaceNone | | ACAPS Application | SADMIN |

9

Customizing Siebel Open UI for Siebel Mobile

This chapter describes how to customize Siebel Open UI for Siebel Mobile and Siebel Mobile disconnected. It includes the following topics:

- [Overview of Customizing Siebel Mobile on page 275](#)
- [Customizing Layout, Views, Menus, Lists, and Controls on page 279](#)
- [Customizing Transitions, Themes, Styles, and Colors on page 322](#)
- [Customizing Scrolling and Swipe on page 336](#)
- [Customizing How Siebel Open UI Interacts with Siebel Mobile Applications on page 343](#)

For information about customizing only Siebel Mobile disconnected, see [Chapter 10, “Customizing Siebel Open UI for Siebel Mobile Disconnected.”](#)

Overview of Customizing Siebel Mobile

This topic includes an overview of how to customize Siebel Mobile. It includes the following information:

- [Mobile Controller and Physical Renderers You Can Modify to Customize Siebel Mobile on page 275](#)
- [Third Party JavaScript Plug-Ins You Can Use to Customize Siebel Mobile on page 276](#)
- [Templates and Style Sheets You Can Use to Customize Siebel Mobile on page 277](#)
- [Setting Up Configuration for Siebel Mobile Examples on page 277](#)
- [Determining Whether or Not Siebel Open UI Is Enabled for Siebel Mobile on page 278](#)

For information about the following items, see *Siebel Connected Mobile Applications Guide*:

- Enabling Siebel Open UI to support Siebel Mobile
- Enabling the Siebel Server for Siebel Mobile
- Enabling the Mobile Web Client to use Siebel Open UI
- Detailed list of Siebel Web Templates, screens, and views that support Siebel Mobile

Mobile Controller and Physical Renderers You Can Modify to Customize Siebel Mobile

You use the JQMLayout mobile controller to control landscape and portrait layout for each type of mobile device.

Table 18 describes the physical renderers you can use with Siebel Mobile. It stores these renderers in the following folder:

`release_number\appsweb\PUBLIC\language_code\release_number\scripts\siebel`

Table 18. Physical Renderers You Can Use with Siebel Mobile

| Renderer | Item Rendered |
|---------------------------|---|
| JQMFormRenderer | Form applet |
| JQMListRenderer | List applet |
| JQMCaListRenderer | Calendar applets for Siebel Mobile |
| JQMGridRenderer | Grid list applet |
| JQMNavBarRenderer | Navigation control |
| JQMMapCtrl | Google map |
| JQMSearchCtrl | Search box |
| JQMPDQPhyRenderer | PDQ dropdown list |
| JQMVisDropdownPhyRenderer | Visibility dropdown list |
| JQMMBMenuRenderer | Applet menu dropdown list |
| JQMScrollContainer | Vertical scrolling in the JQMListRenderer and JQMGridRenderer |
| JQMMsgBarRenderer | Message bar |

Third Party JavaScript Plug-Ins You Can Use to Customize Siebel Mobile

Siebel Mobile supports the following third-party plug-ins:

- **Mobiscroll 2.0.** Add a date and time picker that uses a wheel scroller. For more information, see the Mobiscroll documentation.
- **jQuery Mobile 1.1.1.** You can use the following plug-ins. For more information about jQuery, see the jQuery documentation:
 - **jQuery UI Google Map 3.0.** Integrate Google maps.
 - **jQuery SwipeButton.js.** Add the swipe delete that an iPhone uses to a list view.
 - **jQuery scrollview.** Add grab-and-drag scrolling.
 - **jQuery.signaturepad.min.js.** Create a signature canvas.

Templates and Style Sheets You Can Use to Customize Siebel Mobile

Siebel Mobile comes predefined with the theme-black.css file. You can modify this style sheet to implement the appearance and behavior that you require for Siebel Mobile. The mobiletheme.js file also controls styling. Siebel Open UI prefixes the name of each web template that Siebel Mobile uses with the following text:

CC

Table 19 describes these templates. It stores these templates in the following folder:

`release_number\ses\siebsrvr\WEBTEMPL\OUI WEBTEMPL`

Table 19. Web Templates that Siebel Open UI Uses for Siebel Mobile

| Web Template | Description |
|--|--|
| CCViewDetailMap_Mobile.swt | Maps page. |
| CCViewDetail_Mobile.swt | Detail page. |
| CCViewDetail_Mobile_RelatedItems.swt | Related items page. |
| CCViewDetail_Mobile_Signature.swt | Signatures page. |
| CCAppletFormMobile.swt | Form page. |
| SIAAppletFormGridLayout_NoMenu_OUI.swt | Signature Capture Applet that Siebel Open UI uses in the Signature page. |
| CCAppletFormMobile - Icon.swt | All parent form applets. |
| CCAppletFormMobile - Icon-NoMenu | All parent form applets. |
| CCFormButtonsTop_OUIMobile_NoMenu.swt | Renders the applet title and buttons without the menu button. |
| CCViewDetail_Order_Mobile.swt | Renders the order screen and the return screen in Siebel Consumer Goods. |

Setting Up Configuration for Siebel Mobile Examples

In this topic you set up the configuration that some of the Siebel Mobile examples in this chapter require.

To set up configuration for Siebel Mobile examples

- 1 Create a shortcut to the Chrome browser:
 - a In Microsoft Windows, choose the Start menu, click Programs, Siebel Client, left-click Siebel Call Center - ENU, and then drag it to the desktop to create a shortcut.
 - b Right-click the shortcut you created [Step a](#), and then click Properties.

- c Modify the value in the Target window of the Properties dialog box so that this shortcut does the following:

- Uses the pharma_mobile.cfg file. For more information, see [“Modifying the Application Configuration File” on page 105](#).

- Uses Chrome. You enter the path to the Chrome executable. For example:

```
D: \sea\cli ent\BIN\si ebel . exe /c d: \sea\cli ent\bi n\enu\pharma_mobi le. cfg /u
user_name /p user_password /b "C: \Program Files
(x86)\Googl e\Chrome\Appl icati on\chrome. exe"
```

This shortcut allows you to simulate a tablet mode in the Chrome browser. Make sure you use this shortcut in the mobile topics that this chapter describes.

The pharma_mobile.cfg file is the application configuration file for the ePharma Siebel Mobile application. You can replace pharma_mobile.cfg in this path with the configuration file that your application uses, as necessary.

- 2 Modify the shortcut that you use to start Siebel Call Center.

This shortcut must allow administration privileges for the Siebel Administrator responsibility. To do this, you add the editseeddata switch to the end of the Target string in the shortcut. For example:

```
D: \sea\cli ent\BIN\si ebel . exe /c d: \sea\cli ent\bi n\enu\uagent. cfg /u user_name /p
user_password /b "C: \Program Files (x86)\Googl e\Chrome\Appl icati on\chrome. exe" /
editseeddata
```

Determining Whether or Not Siebel Open UI Is Enabled for Siebel Mobile

This topic includes code that you can use to determine whether or not Siebel Open UI is enabled for Siebel Mobile. You can use this code at run-time to make sure Siebel Open UI is enabled for Siebel Mobile before Siebel Open UI runs any of your customizations that affect Siebel Mobile.

To determine whether or not Siebel Open UI is enabled for Siebel Mobile

- Do one of the following:

- In JavaScript code that runs on the client, you can use the following code:

```
if (Siebel App. S_App. IsMobileApplication() === "true")
```

For more information, see [“IsMobileApplication Method” on page 487](#).

- In a Siebel Web Template that Siebel Open UI runs on the Siebel Server, you can use IsMobileApplicationMode in a swe:case condition. For example:

```
<swe: swi tch>
<swe: case condi ti on="Web Engine State Properti es, IsMobileAppl icati onMode">
<di v id="tbm_1" cl ass="Tool barButton">
<swe: tool bar name="GoOffl ine" >
<swe: tool bari tem property="FormattedHtml "/>
```

```
</swe: tool bar>
</di v>
</swe: case>
<swe: default t>
<di v class="Tier2Tool barContai ner">
<swe: pagei tem id="21">
<di v class="PageI tem"><swe: thi s property="FormattedHtml " /></di v>
</swe: pagei tem>
</di v>
</swe: default t>
</swe: swi tch>
```

Customizing Layout, Views, Menus, Lists, and Controls

This topic describes how to configure layout, views, menus, lists, and controls. It includes the following information:

- [Customizing the Layout for Mobile Devices on page 279](#)
- [Configuring Views to Use Landscape or Portrait Layout on page 282](#)
- [Configuring Siebel Open UI to Display High Interactivity Views in Mobile Web Clients on page 286](#)
- [Using Siebel Web Templates to Modify Siebel Mobile Views on page 288](#)
- [Customizing Menus and Menu Items on page 291](#)
- [Customizing the Number of Columns in Mobile Applets on page 292](#)
- [Customizing the Number of Columns in Mobile Tables on page 295](#)
- [Customizing Mobile Lists on page 299](#)
- [Customizing Tiles on page 303](#)
- [Adding Toggle Controls on page 317](#)
- [Configuring Siebel Open UI to Toggle Row Visibility on page 320](#)
- [Adding the Show More Button to Your Custom Form Applets on page 320](#)

Customizing the Layout for Mobile Devices

This topic describes how to customize the layout according to the mobile device that the user uses, such as a tablet, mobile phone, WebOS, and so on.

To customize the layout for mobile devices

- 1 Open the .css file from one of the following locations:

- The Siebel Server installation folder:

`ORACLE_HOME\WEBMASTER\files\language_code`

- The Siebel client installation folder:

`ORACLE_HOME\PUBLIC\language_code\FILES`

- The Siebel Tools installation folder:

`ORACLE_HOME\PUBLIC\language_code\FILES`

- 2 Add code to the css file you opened in [Step 1](#) that uses media query to create device specific styles.

For more information, see the topic about CSS media queries at the Mozilla Developer Network at https://developer.mozilla.org/en-US/docs/CSS/Media_queries.

For example, add the following code to create styles for tablet or mobile phone:

```
/* iPad */
@media all and (min-device-width: 481px) and (max-device-width: 1024px) and
(orientation: landscape)
{
}
@media all and (min-device-width: 481px) and (max-device-width: 1024px) and
(orientation: portrait)
{
}
/* iPhone */
@media all and (max-device-width: 480px) and (orientation: landscape)
{
}
@media all and (max-device-width: 480px) and (orientation: portrait)
{
}
```

- 3 Identify the .swt file that you must modify, and then open it for editing.

For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

- 4 Add the following tags to the .swt file:

- swe:if
- swe:switch
- swe:case
- swe:default

To add these tags you use the following code:

```
<swe:switch>
  <swe:case condition="Web Engine State Properties, CheckMobileDevice,
    'Device: device_type' ">
    <div id="scrnbar" landscape="Default" portrait="Hide">
    <div id="content" landscape="Default" portrait="Show">
  </swe:case>
<swe:default>
  <swe:switch>
```



```
<swe:case condition="Web Engine State Properties, CheckMobileDevice,
'Device: device_type' ">
  <div id="scrnbar" landscape="Show" portrait="Hide">
    <div id="content" landscape="Hide" portrait="Show">
  </swe:case>
  ...
</swe:switch>
</swe:default>
</swe:switch>
```

where:

- *device_type* is a device that Siebel Open UI supports. The SupportedMobileBrowser server parameter identifies these devices. For more information, see [“Specifying the Supported Mobile Browser” on page 282](#).

For example, the following code sets the layout for a tablet or mobile phone:

```
<swe:switch>
  <swe:case condition="Web Engine State Properties, CheckMobileDevice,
'Device: iPad' ">
    <div id="scrnbar" landscape="Default" portrait="Hide">
    <div id="content" landscape="Default" portrait="Show">
  </swe:case>
  <swe:default>
    <swe:switch>
      <swe:case condition="Web Engine State Properties, CheckMobileDevice,
'Device: iPhone' ">
        <div id="scrnbar" landscape="Show" portrait="Hide">
        <div id="content" landscape="Hide" portrait="Show">
      </swe:case>
      ...
    </swe:switch>
  </swe:default>
</swe:switch>
```

- 5 Set the following parameter in the InfraUIFramework section of the application configuration file so that Siebel Open UI supports the device type that you specify in [Step 4](#):

SupportedMobileBrowser = *device_type_1, device_type_n*

For example:

SupportedMobileBrowser = Pad, iPhone, iPod, Android, webOS

For more information, see [“Modifying the Application Configuration File” on page 105](#).

- 6 Make sure Siebel Open UI is enabled for Siebel Mobile.

For more information, see [“Determining Whether or Not Siebel Open UI Is Enabled for Siebel Mobile” on page 278](#).

Specifying the Supported Mobile Browser

You can use the SupportedMobileBrowser parameter to identify the devices that Siebel Open UI loads in the JQM (jQuery Mobile) view. It uses the following default string value. A comma separates each value that the string contains:

```
SupportedMobileBrowser = iPad,iPhone,iPod,Android-Chrome-Mobile,Android-Chrome,webOS
```

If you do not specify this parameter in the Server Manager or in the application configuration file, then Siebel Open UI uses this default value. You can append values to add more devices. You must append a unique combination of the user agent. For example, to add a parameter for Android Xoom, you use the following string:

```
Android-Xoom
```

You use the following code:

```
SupportedMobileBrowser = iPad,iPhone,iPod,Android-Chrome-Mobile,Android-Chrome,Android-Xoom,webOS
```

For more information, see [“Modifying the Application Configuration File” on page 105](#).

Configuring Views to Use Landscape or Portrait Layout

This topic describes how to configure a view to use landscape or portrait layout. It uses the Opportunity List View as an example.

To configure views to use landscape or portrait layout

- 1** Identify the view that you must modify.
 - a** Log in to the client, and then navigate to the Opportunities screen.
 - b** Click Opportunities List.
 - c** Click Help, and then click About View.
 - d** Make a note of the view name.
For example, Opportunity List View.
- 2** Identify the Web template file that the view that you identified in [Step 1](#) uses:
 - a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click View.
 - c** In the Views list, query the name property for Opportunity List View.
 - d** In the Object Explorer, expand the View tree, and then click View Web Template.

- e In the View Web Templates list, make a note of the value that the Web Template property contains.

For example:

View Detail (Parent with Pointer)

- f In the Object Explorer, click Web Template.
- g In the Web Templates list, query the Name property for the web template name.

For example, query the Name property for the following value. If the name includes a special character, such as a parenthesis, then you must use double quotation marks to enclose the entire value:

"View Detail (Parent with Pointer)"

- h In the Object Explorer, expand the Web Template tree, and then click Web Template File.
- i In the Web Template Files list, make a note of the value that the Filename property contains.
This value identifies the name of the Web template file that you must modify. For example, CCViewDetail_ParentPntr.swt.

3 Modify the Web template file:

- a Choose the application-level View menu, Windows, and then click Web Templates Window.
- b Scroll down in the Web Templates window to locate, and then click the following file:

CCViewDetail_ParentPntr

If you click CCViewDetail_ParentPntr, then Siebel Tools displays the HTML code that the CCViewDetail_ParentPntr.swt file contains. It displays this code in a separate window.

- c Right-click the window that displays the HTML code, and then choose Edit Template.
- d Locate the div element that you must modify.
- e Modify the code.

For example, add the following code:

```
<div id="content1" landscape="Default" portrait="Show">  
<div id="content2" landscape="Default" portrait="Hide">
```

For more information, see ["Landscape and Portrait Tags You Can Add to Div Elements" on page 284](#).

4 Test your modifications.

Landscape and Portrait Tags You Can Add to Div Elements

Table 20 describes the attributes that you can add to a div element that the Web template file contains. Siebel Open UI uses this configuration to determine the orientation when it loads a view or if the user modifies the orientation. It processes each tagged div element for hide or display according to the attribute value for the landscape and portrait layout. It does not process any div element that includes an invalid value for the landscape or portrait tags.

Table 20. Landscape and Portrait Tags You Can Add to Div Elements

| Attribute | Value | Description |
|-----------|---------|--|
| landscape | Default | <p>If the view is in landscape layout, then Siebel Open UI displays the div element. It uses the default css style to determine how to display the div element.</p> <p>To allow the user to toggle between landscape and portrait layout, you add the following css class to the div element. Siebel Open UI toggles the display every time the user clicks the control:</p> <p>toggle</p> |
| landscape | Show | <p>If the view is in landscape layout, then Siebel Open UI displays the div element. Siebel Open UI toggles the display every time the user clicks the control.</p> |
| landscape | Hide | <p>If the view is in landscape layout, then Siebel Open UI hides the div element. Siebel Open UI toggles the display every time the user clicks the control.</p> |
| portrait | Default | <p>If the view is in portrait layout, then Siebel Open UI displays the div element. It uses the default css style to determine how to display the div element.</p> <p>To allow the user to toggle between landscape and portrait layout, you add the following css class to the div element. Siebel Open UI toggles the display every time the user clicks the control:</p> <p>toggle</p> |
| portrait | Show | <p>If the view is in portrait layout, then Siebel Open UI displays the div element. Siebel Open UI toggles the display every time the user clicks the control.</p> |
| portrait | Hide | <p>If the view is in portrait layout, then Siebel Open UI hides the div element. Siebel Open UI toggles the display every time the user clicks the control.</p> |

Configuring Web Templates to Use Landscape or Portrait Layout

The example in this topic configures a web template to use landscape or portrait layout.

To configure web templates to use landscape or portrait layout

1 Modify the web template:

a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

b Choose the application-level View menu, Windows, and then click Web Templates Window.

c Scroll down in the Web Templates window to locate, and then click the following file:

CCPageContainer

d Right-click the window that displays the HTML code, and then choose Edit Template.

e Add the following landscape tag:

```
<swe: frame type="Screenbar" htmlAttr="landscape='Default' portrait='Hide'
marginheight='0' marginwidth='0' noresize scrolling='Auto' ">
  <swe: include file="CCFrameScreenbar.swt"/>
</swe: frame>
```

f Add the following portrait tag:

```
<swe: frame type="Content" htmlAttr="landscape='Default' portrait='Show'
marginheight='0' marginwidth='0' noresize scrolling='Auto' ">
  <swe: include file="CCFrameContentHI.swt"/>
</swe: frame>
```

2 Modify the cascading style sheet:

a Use a text editor to open the theme-blue.css file.

b Add the following code:

```
@media all and (orientation:landscape)
{
  #_swescrnbar {
    width: 20%;
    float: left;
  }
  #_swecontent {
    width: 77%;
    float: right;
  }
}

@media all and (orientation:portrait)
{
  #_swecontent {
    width: 98%;
    float: none;
  }
  #_swescrnbar{
    width: 98%;
    float: none;
  }
}
```

- If any div element is hidden in landscape mode or portrait mode, then you must create a toggle class. Do the following work:

- Locate the proper media query in the theme-blue.css file.
- Create a toggle class in the query you located in [Step o](#) for the div element. Use one of the following tags:

```
landscape = 'Default'  
portrait = 'Default'
```

This configuration allows Siebel Open UI to call the toggle layout. In this example, no hidden div element exists, so it is not necessary to create a toggle class for `#_swescrnbar` or `#_swecontent` for landscape layout.

Configuring a Nested Tag with Landscape or Portrait Layout

Siebel Open UI queries all div elements that contain landscape or portrait attributes to determine hide or display configuration. It does this for all div elements that exist in the DOM. Siebel Open UI allows the following configuration:

- If the parent div element is displayed, then the child div element of this parent can be displayed or hidden.
- If the parent div element is not displayed, then the child div element of this parent must be hidden. It cannot be displayed.

You must make sure that any tags you add use this configuration. For example, consider the following example code:

```
<div id="content" landscape="Default" portrait="Hide">  
  <div id="button"/>  
  <div id="view">  
    <div id="applet1" landscape="Hide" portrait="Show"/>  
    <div id="applet2"/>  
  </div>  
</div>
```

This code does the following:

- **Landscape layout.** Displays the button and applet2, but hides applet1.
- **Portrait layout.** Hides the button, applet2, and applet1.

Configuring Siebel Open UI to Display High Interactivity Views in Mobile Web Clients

This topic describes how to configure Siebel Open UI to display a high interactivity view in the Mobile Web Client.

To configure Siebel Open UI to display high interactivity views in Mobile Web Clients

1 Modify the screen:

- a** Open Siebel Tools using the Siebel Mobile tag.
For more information, see *Using Siebel Tools*.
- b** In the Object Explorer, click Screens.
- c** In the Screens list, query the Name property for ePharma Account Mobile.
- d** In the Object Explorer, expand the Screen tree, and then click Screen View.
- e** In the Screen Views list, query the Name property for Pharma Account Contact View - Mobile.
- f** Choose the Edit menu, and then click Copy Record.
- g** Modify the following properties of the copy of the screen view that you created in [Step f](#).

| Property | Value |
|-------------------------------|---|
| View | TNT SHM Intermediary Account Opportunities View |
| Sequence | 7 |
| Viewbar Text -String Override | Opportunities |
| Menu Text -String Override | Opportunities |

- h** Compile your modifications.

2 Test your modifications:

- a** Use the shortcut that you modified in [Step c on page 278](#) to open the ePharmacy application.
Siebel Open UI displays the ePharmacy application in desktop mode.
- b** In Chrome, click the Preferences menu, and then the Advanced menu item.
- c** Make sure the following option includes a check mark, and then close the dialog box:
Show Developer Menu in Menu Bar
- d** Click the menu bar, click Develop, click User Agent, and then choose iPad.
To use the iPad simulation mode, you can also click the mask icon, and then choose iPad.
Chrome displays the mask icon to the left of the wrench icon in the upper-right corner of the Chrome browser.

- e Click the Accounts screen, and then click the Opportunities tab.

Siebel Open UI displays the view you modified. Make sure it resembles the following view. This view is not converted for use in Siebel Mobile so Siebel Open UI does not display the mobile user interface. To apply this user interface, see [“Using Siebel Web Templates to Modify Siebel Mobile Views” on page 288](#).



The screenshot shows the Siebel Open UI Accounts screen. On the left is a navigation pane with icons for Accounts, Contacts, and other functions. The main area contains a form for 'Morris Plains' with fields for 'Country' (USA), 'User Email' (9734495670), and 'Zip Code' (07950). Below the form is a list of tabs: Contact, Calls, Addresses, Attentions, Relationships, and Personalization.

- f Close the ePharmacy application.

Using Siebel Web Templates to Modify Siebel Mobile Views

This topic describes how to apply a Siebel Web Template (SWT) to a Siebel Mobile view.

To use Siebel Web Templates to modify appearance and behavior

- 1 Configure Siebel Open UI to display the desktop view that you must modify in the Mobile Web Client.

For more information, see [“Configuring Siebel Open UI to Display High Interactivity Views in Mobile Web Clients” on page 286](#).

- 2 Modify the view:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b In the Object Explorer, click View.

- c** In the Views list, query the name property for the following value:

TNT SHM Intermediary Account Opportunities View

- d** Make a copy of the record that you located in [Step c](#).

- e** Modify the following properties of the view that you created in [Step d](#).

| Property | Value |
|-----------------|--|
| Name | TNT SHM Intermediary Account Opportunities View - Mobile |
| Project | Mobile LS |
| Business Object | Account |

- f** In the Object Explorer, expand the View tree, and then click View Web Template.

- g** In the View Web Templates list, modify the following property Base view web template.

| Property | Value |
|--------------|----------------------------------|
| Web Template | View Detail Mobile Related Items |

- h** In the Object Explorer, expand the View Web Template tree, and then click View Web Template Item.

- i** In the View Web Template Items list, query the Name property for Opportunity List Applet.

- j** Modify the following property of the Opportunity List Applet view web template item.

| Property | Value |
|-----------------|-------|
| Item Identifier | 4 |

- k** In the View Web Template Items list, query the Name property for SIS Account Entry Applet.

- l** Modify the following properties SIS Account Entry Applet view web template item.

| Property | Value |
|----------|-----------------------------------|
| Applet | Mobile Pharma Account Form Applet |
| Name | Mobile Pharma Account Form Applet |
| Mode | Base |

- m** In the Object Explorer, click View, and then modify the following properties of the TNT SHM Intermediary Account Opportunities View.

| Property | Value |
|---------------|-----------------------------------|
| Thread Applet | Mobile Pharma Account Form Applet |

3 Modify the screen:

- a** In the Object Explorer, click Screen.
- b** In the Screens list, query the Name property for ePharma Account Mobile.
- c** In the Object Explorer, expand the Screen tree, and then click Screen View.
- d** In the Screen Views list, query the Name property for the following value:
TNT SHM Intermediary Account Opportunities View
- e** Modify the following property of the TNT SHM Intermediary Account Opportunities View - Mobile.

| Property | Value |
|----------|--|
| View | TNT SHM Intermediary Account Opportunities View - Mobile |

4 Compile your modifications.

5 Configure the responsibilities:

- a** Use the shortcut that you modified in [Step 2 on page 278](#) to open Siebel Call Center.
- b** Navigate to the Administration - Application screen, and then click the Responsibilities link.
- c** In the Responsibilities list, query the Responsibility field for Siebel Administrator.
- d** In the Views list, add the following view:

TNT SHM Intermediary Account Opportunities View - Mobile

6 Test your modifications:

- a** Use the shortcut that you modified in [Step c on page 278](#) to open the ePharmacy application.

- b Click the Accounts screen, and then click the Opportunities tab.

Siebel Open UI displays the view you modified. Make sure it resembles the following view.



Customizing Menus and Menu Items

This topic describes how to customize menus and menu items.

Customizing Menus

This topic describes how to customize a menu button for an applet.

To customize menus

- Modify the CCListButtonsTop_Mobile.swt file.

Customizing Menu Items

This topic describes how to customize the menu items that a menu displays. Siebel Mobile uses a list applet to display a menu. It restricts the height of each menu item. The user can scroll up or down in the menu to access each menu item. You can modify an applet method menu item or a class method menu item.

To customize menu items

- 1 Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- 2 Optional. Modify the applet method menu item:

- a In the Object Explorer, click Applet.
- b In the Applets list, query the Name property for the name of the list applet that Siebel Open UI uses to display the menu.
- c In the Object Explorer, expand the Applet tree, and then click Applet Method Menu Item.
- d Add an applet method menu item.

- 3 Optional. Modify the class that the applet uses:

- a In the Object Explorer, click Class.
- b In the Class list, query the Name property for CSSSWEFrame or CSSSWEFrameList.
- c In the Object Explorer, expand the Class tree, and then click Class Method Menu Item.
- d Add a class method menu item.

For more information about setting the properties for an applet method menu item or a class method menu item, see *Configuring Siebel Business Applications*.

Customizing the Number of Columns in Mobile Applets

Siebel Open UI comes predefined to display a maximum of four list columns in a Siebel Mobile phone client, and eight list columns in a tablet. This topic describes how to write a custom renderer that overwrites this number of maximum columns. You can customize the number of columns at the following different levels for tablets and for phones:

- **Applet level.** Maximum number of list columns.
- **Globally.** Default maximum number of list columns. The custom renderer modifies only the global configuration.

To customize the number of columns in mobile applets

- 1 Modify the applet:

- a Open Siebel Tools.
For more information, see *Using Siebel Tools*.
- b In the Object Explorer, click Applet.
- c In the Applets list, locate the applet that you must modify.
- d In the Object Explorer, expand the Applet tree, and then click Applet User Prop.

- e In the Applet User Properties list, add the following applet user properties.

| Name | Value |
|-------------------------|---|
| Max List Columns Phone | Enter a number. For example, to display a maximum of five columns in a phone, enter the number 5. |
| Max List Columns Tablet | Enter a number. For example, to display a maximum of seven columns in a tablet, enter the number 7. |
| ClientPMUserProp | <p>Enter the following value:</p> <p>Max Li st Co l um ns Ph one, Ma x Li st Co l um ns Ta bl et</p> <p>You add a series of applet user property names. Use a comma to separate each name. Siebel Open UI sends these applet user properties from the Siebel Server to the client at run time. In this example, you add the other applet user properties that you specify in this step.</p> |

- f Compile your modifications.

2 Add custom physical renderers.

Siebel Open UI comes predefined to use the jqmli strenderer.js file for the phone and the jqmtabletli strenderer.js file for the tablet to determine the maximum number of columns. Siebel Open UI derives the jqmtabletli strenderer.js file from the jqmli strenderer.js file. You can create a custom physical renderer to overwrite the SetDefaultMaxListColumns method that these files specify. Do the following:

- a Create a JavaScript file in the following folder:

scri pts\si ebel \custom

You must use this folder. You can use any file name. For this example, use custom_phone_renderer.js. For more information, see [“Organizing Files That You Customize” on page 122](#).

- b Add the following code to the file you created in [Step a](#). This code overwrites the methods that Siebel Open UI uses to render columns in a phone:

```
if (typeof (Siebel AppFacade. JQMPhoneLi stRenderer) ===
"undefined") {
    Siebel JS. Namespace(' Siebel AppFacade. JQMPhoneLi stRenderer' );
    define("siebel /j qmphoneli strender", ["order! 3rdParty/j qmobi le/
mobiscrol l. custom-2. 5. 0. mi n",
    "order! 3rdParty/j qmobi le/j query. swi peButton. mi n", "order! 3rdParty/j qmobi le/
j query. easi ng. 1. 3",
    "order! 3rdParty/j qmobi le/j query. mobi le. scrol lvi ew", "order! 3rdParty/
j query. mobi le. scrol lvi ew-ext",
    "order! siebel /j qmsearchctrl ", "order! siebel /j qmformrender", "order! siebel /
jqmscrol lcontai ner", "order! siebel /j qml i strender"],
    function () {
        Siebel AppFacade. JQMPhoneLi stRenderer = (function() {
            function JQMPhoneLi stRenderer(pm)
```

```
{Siebel AppFacade.JQMPhoneListRenderer.superclass.constructor.call(
    this, pm);
}
Siebel JS.Extend(JQMPhoneListRenderer, Siebel AppFacade.JQMListRenderer);
JQMPhoneListRenderer.prototype.SetDefaultMaxListColumns = function() {
    this.m_defaultMaxListColumns = 5;
    //overwritten value
    for phone maximum list columns
};
return JQMPhoneListRenderer;
};
})();
return "Siebel AppFacade.JQMPhoneListRenderer";
});
}
```

- c** Create a new JavaScript file named `custom_tablet_renderer.js` in the following folder:

`CLIENT_HOME\PUBLIC\language_code\bui\id_number\scripts\siebel\custom`

You must create a separate file for each platform.

3 Configure the manifest:

- a** Log in to the Siebel client with administrative privileges.

For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).

- b** Navigate to the Administration - Application screen, and then the Manifest Files view.
- c** In the Files list, add the following file.

| Field | Value |
|-------|--------------------------------|
| Name | siebel/jqmphonelistrenderer.js |

You add the custom files that you created in [Step 2](#).

- d** Navigate to the Administration - Application screen, and then the Manifest Administration view.
- e** In the UI Objects list, specify the following applet.

| Field | Value |
|------------|--|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Specify the name of the applet that you modified in Step 1 . |

- f** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a phone platform.

| Field | Value |
|------------|-------|
| Expression | Phone |
| Level | 1 |

- g** In the Files list, add the following file:

`si ebel /j qmphonel i strenderer. j s`

- h** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a tablet platform.

| Field | Value |
|------------|--------|
| Expression | Tablet |
| Level | 2 |

- i** In the Files list, add the following file:

`si ebel /j qmtabl etl i strenderer. j s`

- j** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 3 |

- k** In the Files list, add the following file:

`si ebel /j qml i strenderer. j s`

- 4** Test your modifications:

- a** Log in to the client through a phone, and then verify that Siebel Open UI displays the maximum number of columns that you specified for the phone.
- b** Log in to the client through a tablet, and then verify that Siebel Open UI displays the maximum number of columns that you specified for the tablet.

Customizing the Number of Columns in Mobile Tables

Siebel Open UI comes predefined to display a maximum of four list columns in a phone and eight list columns in a tablet. You can use applet user properties or JavaScript to modify the number of columns that Siebel Open UI displays.

Siebel Open UI uses the JQMListRenderer physical renderer to render an applet in a telephone. The *JQMGridRenderer* is a list physical renderer that renders controls in a table, such as a combo box, pick list, check box, date and time control, or text control. It is read-only, by default. However, the user can click an editable cell in a table to update the value in this cell. You can configure the manifest so that Siebel Open UI uses the JQMGridRenderer instead of JQMListRenderer. For more information, see “Configuring Manifests” on page 128.

Figure 33 includes an example of the Contacts list that the JQMTabletGridRenderer renders in a tablet. The *JQMTabletGridRenderer* is a physical renderer that Siebel Open UI uses to render an applet in a tablet.

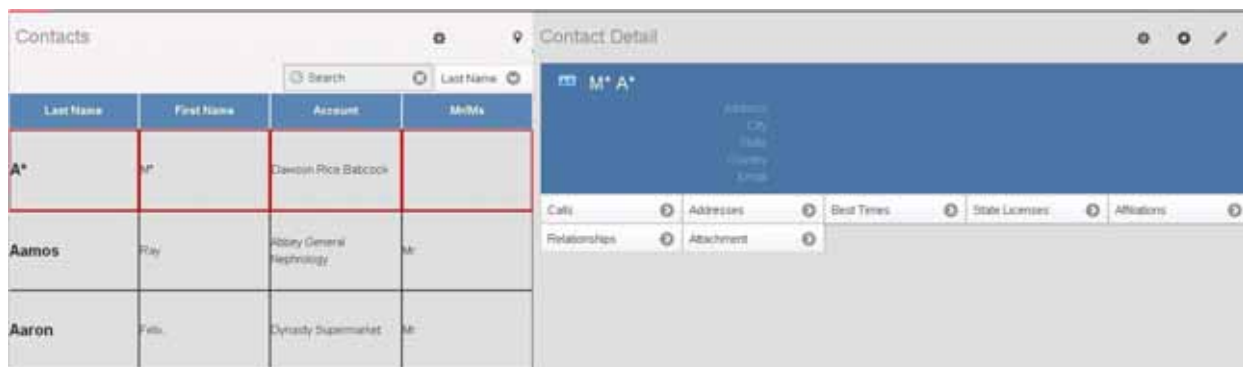


Figure 33. Example of a List That the JQMTabletGridRenderer Renders in a Tablet

Using Applet User Properties to Modify the Number of Columns in Tables

This topic describes how to use applet user properties to modify the number of columns that Siebel Open UI displays in a table that it renders in a list applet.

To use applet user properties to modify the number of columns in tables

- 1** Modify the applet:
 - a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click Applet.
 - c** In the Applets list, query the Name property for the applet that contains the table that you must modify.
 - d** In the Object Explorer, expand the Applet tree, and then click Applet User Prop.

- e In the Applet User Properties list, add the following applet user properties.

| Name | Value |
|-------------------------|--|
| Max List Columns Phone | <p>Enter a numeric value. For example, enter the following value to configure Siebel Open UI to display three columns for this list applet:</p> <p>3</p> <p>This applet user property controls the number of columns that Siebel Open UI displays in a list applet that it renders in a phone.</p> |
| Max List Columns Tablet | <p>Enter a numeric value. For example, enter the following value to configure Siebel Open UI to display six columns for this list applet:</p> <p>6</p> <p>This applet user property controls the number of columns that Siebel Open UI displays in a list applet that it renders in a tablet.</p> |
| ClientPMUserProp | <p>Specify the list of user properties that Siebel Open UI sends to the presentation model that it uses for the applet. For this example, you specify the other applet user properties that you added in this step:</p> <p>Max Li st Col umns Phone, Max Li st Col umns Tabl et</p> |

- f Compile your modifications.

2 Configure the manifest:

- a Log in to the Siebel application.
- b Navigate to the Administration - Application screen.

For more information about the screens you use in this step, see [“Configuring Manifests” on page 128](#).

- c Click Manifest Administration.
- d In the UI Objects list, add the following UI applet.

| Field | Value |
|------------|-------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | My Applet |

This step defines the user interface object for My Applet.

- e In the Object Expression list, add the following expressions. Use the File list to add each file.

| Expression | Level | File |
|------------|-------|---------------------------------|
| Phone | 1 | siebel/jqmgridrenderer.js |
| Tablet | 2 | siebel/jqmtabletgridrenderer.js |
| Desktop | 3 | siebel/jqmgridrenderer.js |

Using JavaScript to Modify the Number of Columns in Tables

This topic describes how to use JavaScript to modify the number of columns that Siebel Open UI displays in a table that it renders in a list applet.

To use JavaScript to modify the number of columns in tables

- 1 Copy one of the following files to your custom folder:
 - **jqmgridrenderer.js**. Copy this file to modify a table that Siebel Open UI displays in a phone.
 - **jqmtabletgridrenderer.js**. Copy this file to modify a table that Siebel Open UI displays in a tablet.

You must copy these files only to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

For more information about where these JavaScript files reside and the folders that you can use to store your customizations, see [“Organizing Files That You Customize” on page 122](#).

- 2 Use a JavaScript editor to open the file that you copied in [Step 1](#).
- 3 Add the following code:

```
if (typeof (Siebel AppFacade. JQMCustomGridRenderer) === "undefined") {
    Siebel JS. Namespace(' Siebel AppFacade. JQMCustomGridRenderer' );
    //Module with its dependencies
    define("siebel /jqmcustomgridrenderer ", ["order! 3rdParty/jqmobi le/
    mobi scrol l. custom-2. 5. 0. mi n", "order! 3rdParty/jqmobi le/
    jquery. swi peButton. mi n", "order! 3rdParty/jqmobi le/
    "order! 3rdParty/jqmobi le/jquery. mobi le. scrol l vi ew", "order! 3rdParty/
    jquery. mobi le. scrol l vi ew-ext", "order! si ebel /jqmsearchctrl ", "order! si ebel /
    jqmformrenderer", "order! si ebel /jqmscrol l contai ner", "order! si ebel /
    jqmgridrenderer"],
    function () {
        Siebel AppFacade. JQMCustomGridRenderer = (function() {
            Siebel JS. Extend(JQMCustomGridRenderer, Siebel AppFacade. JQMGri dRenderer);
            JQMCustomGridRenderer. prototype. SetDefaul tMaxLi stCol umns = function() {
                thi s. m_defaul tMaxLi stCol umns = 3;
            };
        });
    });
}
```

- 4 Save your modifications.
- 5 Add the physical renderer that you modified in [Step 3](#) to the manifest:

- a Log in to the Siebel application.
- b Navigate to the Administration - Application screen, and then the Manifest Files list.
- c In the Files list, add the following file.

| Field | Value |
|-------|---------------------------------|
| Name | siebel/jqmcustomgridrenderer.js |

- d Do [Step 2 on page 297](#), and also add the siebel/jqmcustomgridrenderer.js file to each platform that you add in [Step e on page 298](#).

Customizing Mobile Lists

This topic describes how to configure Siebel Open UI to create a custom mobile list.

To customize mobile lists

- 1 Configure the manifest:
 - a Log in to the Siebel client with administrative privileges.
For more information about the screens that you use in this step, see [“Configuring Manifests” on page 128](#).
 - b Navigate to the Administration - Application screen, and then the Manifest Files view.
 - c In the Files list, add the following file.

| Field | Value |
|-------|---------------------------------|
| Name | siebel/jqmcustomlistrenderer.js |
| Name | siebel/jqgridrender.js |

- d Navigate to the Administration - Application screen, and then the Manifest Administration view.
- e In the UI Objects list, specify the following applet.

| Field | Value |
|------------|--|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | LS Pharma Account Calls List Applet - Mobile |

- f** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a mobile platform.

| Field | Value |
|------------|--------|
| Expression | Mobile |
| Level | 1 |

- g** In the Files list, add the following file:

`si ebel /j qmcustoml i strenderer. j s`

- h** In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on a desktop platform.

| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 2 |

- i** In the Files list, add the following file:

`si ebel /j qgri drenderer. j s`

2 Configure the list renderer:

- a** Download the jqmlistrender.js file to the following folder:

`CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom`

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

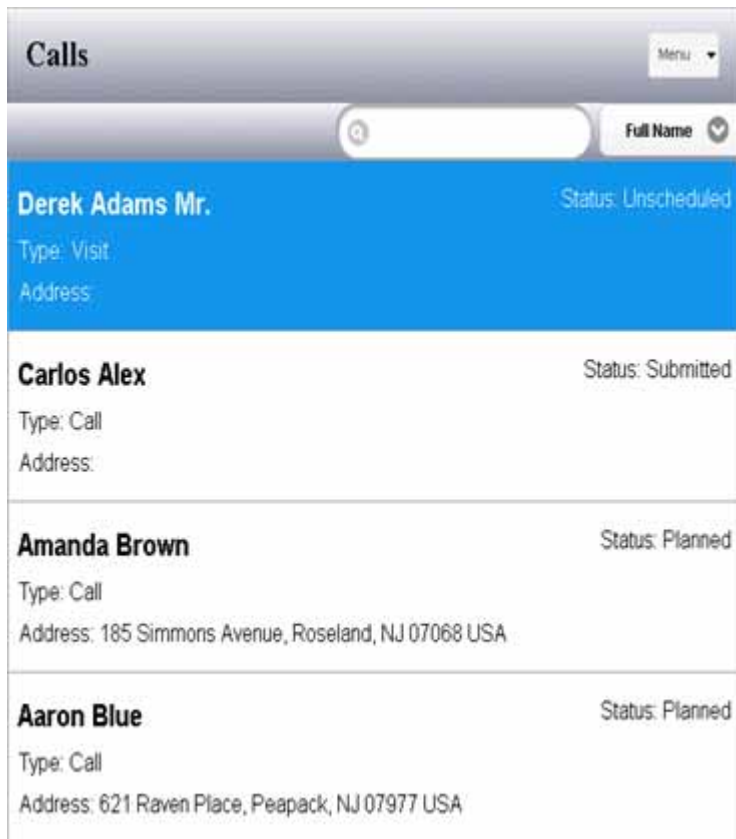
- b** Rename the file you downloaded in [Step a](#) to jqmcustomlistrender.js.
- c** Use a JavaScript editor to open the jqmcustomlistrender.js file.

- d** Modify the following code that resides in the `jqmcustomlistrender.js` file. Bold font indicates the code that you must modify.

| Replace This Old Code. . . | . . .With This New Code |
|--|--|
| <pre>Si ebel App. S_App. Regi sterConstruct orAgai nstKey(Si ebel App. Constants. get("SWE_UI DEF_LI ST_PRENDR"), "Si e bel AppFacade. JQMCustomLi stRendere r");</pre> | <pre>defi ne("si ebel /j qmcustoml i strender", ["order! 3rdParty/j qmobi l e/ mobi scrol l . custom-2. 5. 0. mi n", "order! 3rdParty/j qmobi l e/ j query. swi peButton. mi n", "order! 3rdParty/ j qmobi l e/j query. easi ng. 1. 3", "order! 3rdParty/j qmobi l e/ j query. mobi l e. scrol l vi ew", "order! 3rdParty/j query. mobi l e. scrol l vi ew- ext", "order! si ebel /j qmsearchctrl ", "order! si ebel /j qmformrender", "order! si ebel /j qmscrol l contai ner"], functi on () {</pre> <p>Use the Define method to identify the physical renderer file. You must use the Define method to make Siebel Open UI can identify the constructor. You must include the relative path and the name of the presentation model file without the file name extension. For more information, see "Define Method" on page 506.</p> |
| <p>Locate the following code in the <code>JQMCustomListRenderer.prototype.ShowUI</code> method:</p> <pre>this.s.GetJQMLi st().html("<ul id=' " + li stId + "'>");</pre> | <pre>this.s.GetJQMLi st().html("<ol id=' " + li stId + "'>");</pre> |
| <pre>JQMLi stRender. prototype. GetLi st Hei ghtreturn \$("#" + this.s.GetScrol l contai ner()). chi l dren("ul "). hei ght();</pre> | <pre>return \$("#" + this.s.GetScrol l contai ner()). chi l dren("ol "). hei ght();</pre> |
| <p>Locate the second occurrence of the following code:</p> <pre>for (index = 0; index < Col Li st. l ength; i ndex++)</pre> <pre>{. . .}</pre> | <p>Delete the entire section.</p> |

- 3** Test your modifications:
- a** Log in to the ePharma application.
 - b** Reset the Safari browser so that it uses mobile mode.
 - c** Navigate to the Calls screen.

- d** Make sure the Calls applet uses the new physical renderer when in mobile mode. This renderer displays calls in the following numbered list.



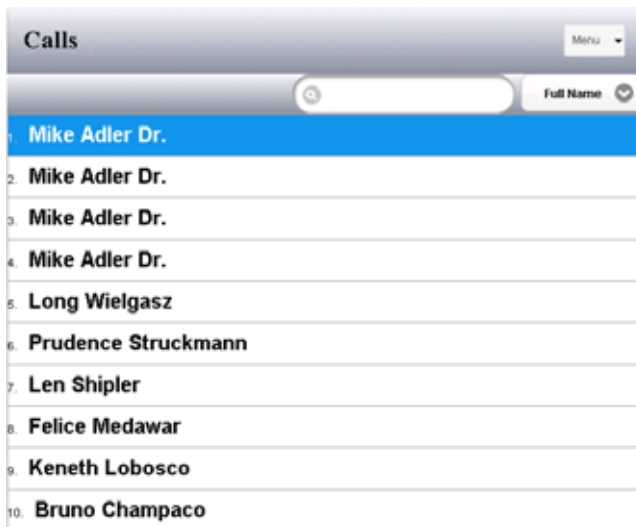
- 4** Modify the style:
 - a** In Windows Explorer, navigate to the following folder:
`CLIENT_HOME\PUBLIC\language_code\files`
 - b** Use an editor to open the theme-black.css file.
 - c** Locate the following code:

```
.ui-li-heading {  
  font-family: Helvetica, Neue;  
  font-size: 15pt; /15pt; / /*100%; */  
  font-weight: bold;  
  display: block;  
  margin: .5em 0 0 .4em;  
  text-overflow: ellipsis;  
  overflow: hidden;  
  white-space: nowrap;  
}
```

- d** Modify the code that you located in [Step c](#). You modify the display from block to inline. See the following bolded code:

```
ui-li-heading {
font-family: Helvetica, Neue;
font-size: 15pt; /15pt; / /*100%; */
font-weight: bold;
display: inline;
margin: .5em 0 0 .4em;
text-overflow: ellipsis;
overflow: hidden;
white-space: nowrap;
}
```

- 5** Test your modifications:
- a** Restart the Safari browser.
 - b** Navigate to the Calls screen.
 - c** Make sure the Calls applet uses the following inline display.



Customizing Tiles

This topic describes how to customize tiles. It includes the following topics:

- [“Overview of Customizing Tile Applets” on page 304](#)
- [“Customizing Tiles for List Applets” on page 306](#)
- [“Customizing Tiles for Mobile Lists” on page 312](#)
- [“Configuring Horizontal and Vertical Scrolling in Tile Applets” on page 314](#)
- [“Adding Controls to Tile Applets” on page 317](#)

Overview of Customizing Tile Applets

Figure 34 includes an example of an applet that Siebel Open UI displays as a set of tiles. A *tile* is a type of user interface object that Siebel Open UI uses to display a Siebel CRM record as a square or rectangle.

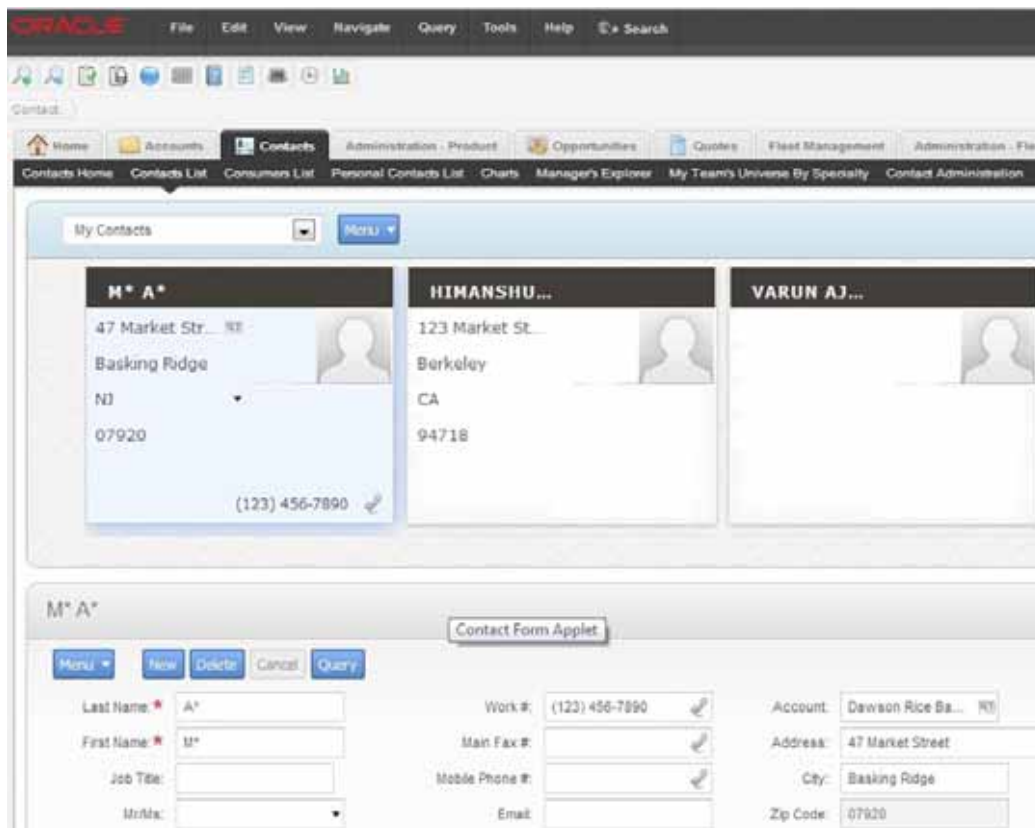


Figure 34. Example Tile Applet

This example includes the following items:

- The Contact List applet and the Contact List View.
- Usage of the cascading style sheet to indicate the chosen tile in blue. You can modify this cascading style sheet to customize color usage.
- Usage of applet controls to display images. These controls determine the pick applet icon, MVG icon, email icon, and other icons that Siebel Open UI displays in the chosen tile. If the user clicks a different tile, then Siebel Open UI hides the icons in the current tile and displays them in the tile that the user clicks. You can modify the applet controls in Siebel Tools to customize how Siebel Open UI displays them. For more information, see ["Adding Controls to Tile Applets" on page 317](#).

Applet Mode Configurations for Tiles

This topic describes two applet modes that Siebel Open UI can use to display a tile applet. You can modify the applet modes in [“Customizing Tiles for List Applets” on page 306](#).

[Figure 35](#) includes an example of an applet that Siebel Open UI displays in the Query applet mode. The user can query records in the applet, but cannot modify and field values.

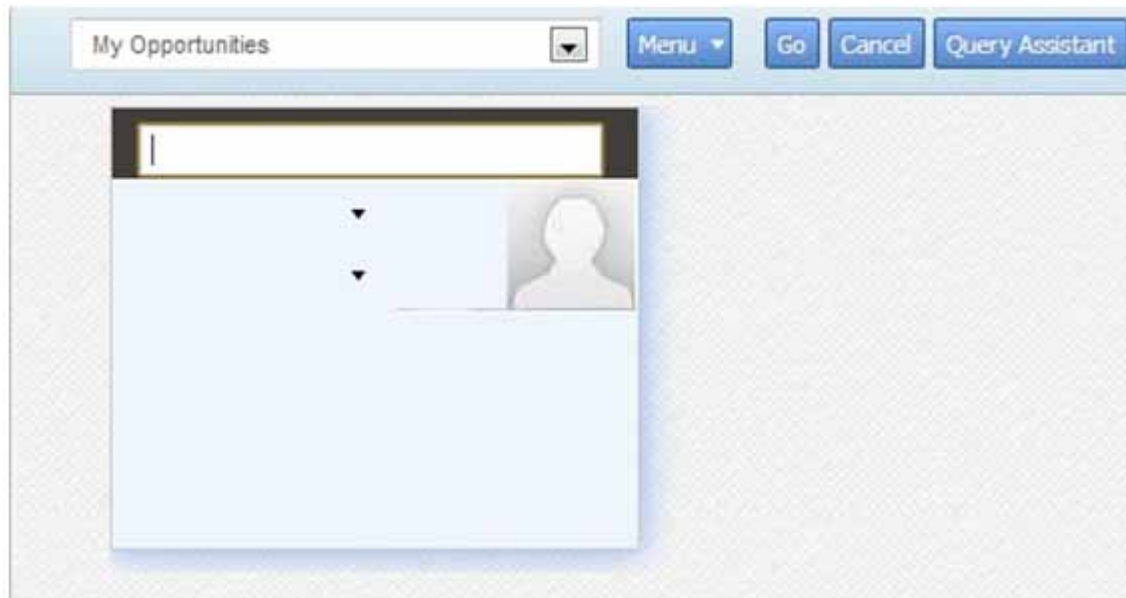


Figure 35. Example Tile Applet in the Query Applet Mode

Figure 36 includes an example of an applet that Siebel Open UI displays in the Edit applet mode. The user can query records in this applet and can also modify field values. Siebel Open UI displays the control field values in each tile as a label, by default. The user clicks the field value to edit it.

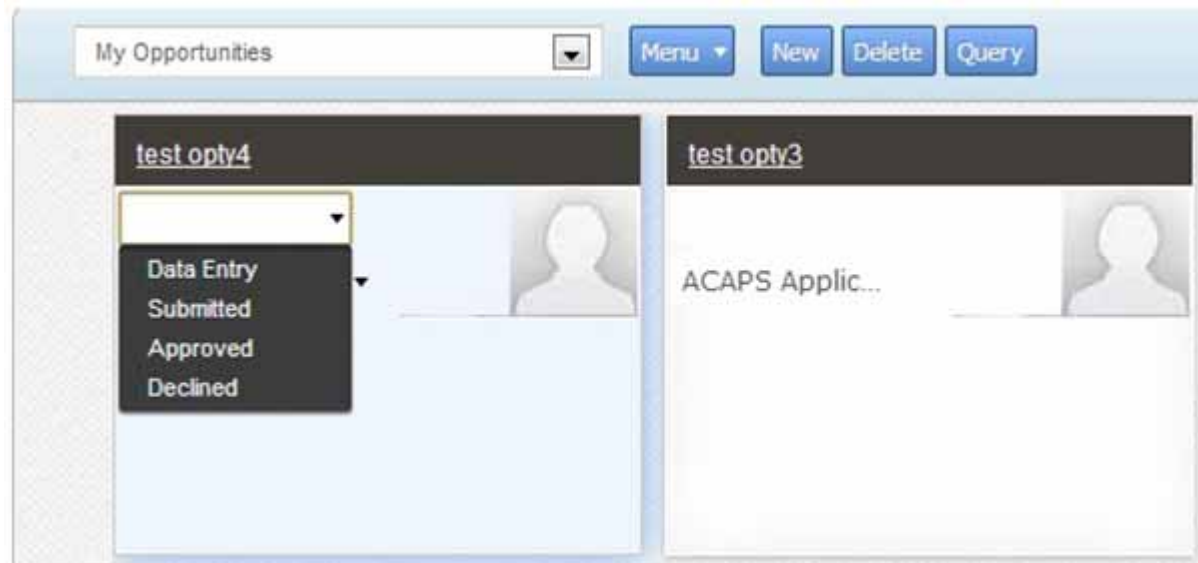


Figure 36. Example Tile Applet in Edit Mode

Customizing Tiles for List Applets

This topic describes how to customize tiles so that Siebel Open UI can display them in different applet modes. To view a tile applet that is similar to the applet that you configure in this topic, see [“Overview of Customizing Tile Applets” on page 304](#). For information about custom tiles in a mobile environment, see [Customizing Tiles for Mobile Lists on page 312](#).

To customize tiles for list applets

- 1** Configure the SWT file so that Siebel Open UI can use it to render tiles:
 - a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b** In the Object Explorer, click Applet.
 - c** In the Applets list, query the Name property for the applet that you must modify.
For example, query for Opportunity List Applet.
 - d** In the Object Explorer, expand the Applet tree, and then click Applet Web Template.

- e In the Applet Web Templates list, choose the record that Siebel Open UI uses to display the applet in the applet mode that your deployment requires.
For example, if you must display the applet in Base mode, then choose the record that includes Base in the Name property.
- f Make a note of the value that the Web Template property contains.
For example, the Web Template property contains the following value for Base mode:
`Applet List (Base/EditList)`
- g In the Object Explorer, click Web Template.
- h Query the Name property for the value that you noted in [Step f](#).
For example, query the Name property for the following value. If the value includes special characters, such as parentheses or a slash, then enclose the value in quotation marks:
`"Applet List (Base/EditList)"`
- i In the Object Explorer, expand the Web Template tree, and then click Web Template File.
- j In the Web Template Files list, make a note of the value that the Filename property contains.
For example, the Filename property contains the following value for the Applet List (Base/EditList) web template:
`CCAppletList_B_EL.swt`
- k In Windows Explorer, navigate to the following folder:
`si ebsrvr\webtempl \oui webtempl`
- l Make a copy of the file that you noted in [Step j](#), and then rename this copy.
For example:
`CCAppletList_B_EL_custom_tiles.swt`
- m Add each of the following tags to the copy that you created in [Step l](#).

| SWE Tag | Description |
|--------------------------------|--|
| swe:this property="NoGrid" | Specifies that the current list layout is not a Grid layout. |
| swe:this property="Horizontal" | Specifies the scroll direction. For more information, see "Configuring Horizontal and Vertical Scrolling in Tile Applets" on page 314. |

For example code that includes these tags, see ["Code That You Can Use to Customize Tiles for List Applets"](#) on page 310.

- n Modify the swe:list tag so that it includes the HTML layout for each tile.
The swe:list tag includes the swe:list-record tag that specifies the layout for each tile. For example code that includes this HTML layout, see the contents of the swe:list tag in ["Code That You Can Use to Customize Tiles for List Applets"](#) on page 310.

2 Create a new web template:

- a** In the Object Explorer, click Web Template.
- b** In the Web Templates list, add the following web template.

| Property | Value |
|----------|------------------------|
| Name | Applet Tile |
| Type | Applet Template - List |

- c** In the Object Explorer, expand the Web Template tree.
- d** In the Object Explorer, click Web Template File.
- e** In the Web Template Files list, add the following web template file.

| Property | Value |
|----------|--|
| Name | Applet Tile |
| Filename | Specify the file that you modified in Step 1 . For this example, you specify the following file: CCAppletList_B_EL_custom_tiles.swt |
| Bitmap | Applet_List |

3 Configure the applet so that Siebel Open UI can display it as a tile layout:

- a** In the Object Explorer, click Applet.
- b** In the Applets list, query the Name property for the applet that you must modify.
For example, query for Opportunity List Applet.
- c** In the Object Explorer, expand the Applet tree, and then click Applet Web Template.
- d** In the Applet Web Templates list, add the following applet web template.

| Property | Value |
|----------|--|
| Name | Enter a name that describes the applet mode. For this example, Siebel Open UI must allow the user to edit values, so enter the following value: Edit List |

| Property | Value |
|--------------|--|
| Type | Choose the applet modes that Siebel Open UI must use to display the applet. For this example, choose the following value: Edit List For more information, see “Applet Mode Configurations for Tiles” on page 305 . |
| Web Template | Choose the following web template that you added in Step 2 : Applet Tile |

4 Configure the manifest:

- a Log in to the Siebel client with administrative privileges.
- b Navigate to the Administration - Application screen, and then the Manifest Administration view. For more information about the screens you use in this step, see [“Configuring Manifests” on page 128](#).
- c In the UI Objects list, specify the following applet.

| Field | Value |
|------------|-------------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | Opportunity List Applet |

- d In the Object Expression list, add the following expression. Siebel Open UI uses this expression to render the applet on the desktop.

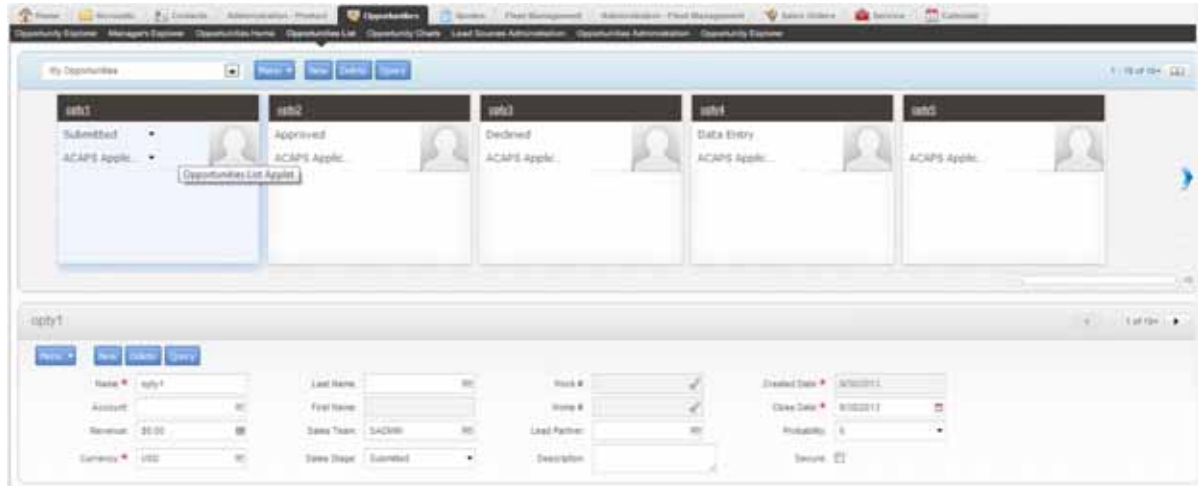
| Field | Value |
|------------|---------|
| Expression | Desktop |
| Level | 1 |

- e In the Files list, add the following file:
siebel/TileScrollContainer.js

5 Test your modifications.

- a Log in to the client, and then navigate to the Opportunities List.

- b** Verify that the client displays a list that resembles the following layout.



Code That You Can Use to Customize Tiles for List Applets

Siebel Open UI comes predefined with the CCAppl etLi st_Ti l e.swt file that resides in the following folder, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4:

```
si ebsrvr\webtempl \oui webtempl
```

It includes an example tile format that you can customize to meet your deployment requirements. It includes the following code. Bold font indicates modifications that customize tiles:

```
<!-- Template Start: CCAppl etLi st_Ti l e.swt -->
<swe: i ncl ude fi l e="CCApl et_NamedSpacer.swt"/>
<swe: control i d="1100">
  <di v cl ass="CmdTxt">
    <swe: thi s property="FormattedHtml " hi ntText="Outsi de Appl et Hel p Text"
    hi ntMapType="Control "/>
  </di v>
</swe: control >
<swe: form>
  <swe: i ncl ude fi l e="CCTi tle_Named.swt"/>
  <di v cl ass="swe: thi s. Sel ectStyl e">
    <swe: swi tch>
      <swe: case condi ti on="Web Engi ne State Propertie s, I sMobi l eAppl i cati onMode">
        <swe: i ncl ude fi l e="CCLi stButtonsTop_Mobi l e.swt"/>
      </swe: case>
      <swe: defaul t>
        <swe: i ncl ude fi l e="CCLi stButtonsTop.swt"/>
      </swe: defaul t>
    </swe: swi tch>
    <swe: error type="Popup">
      <di v cl ass="swe: cl ass Appl etBack">
        <di v cl ass="error">
          <swe: thi s property="FormattedHtml "/>
        </di v>
      </swe: error>
    </swe: error>
  </di v>
</swe: form>
```

```

</div>
</swe:error>
<div class="AppletHListBorder siebui-tile-container">
  <swe: this property="NoGrid"/>
  <swe: this property="Horizontal"/>
  <swe: list>
    <swe: list-record>
      <div class="siebui-tile-name">
        <swe: for-each count="3" startValue="500" iteratorName="currentId">
          <swe: control id="swe: currentId" hintMapType="FormItem">
            <div class="siebui-form-data" align="swe: this.TextAlignment">
              <swe: this property="FormattedHtml" hintText="Field"/>
            </div>
          </swe: control>
        </swe: for-each>
      </div>
      <div class="siebui-tile-details-row1">
        <ul>
          <swe: for-each count="4" startValue="510" iteratorName="currentId">
            <li>
              <swe: control id="swe: currentId" hintMapType="FormItem">
                <div class="siebui-form-data" align="swe: this.TextAlignment">
                  <swe: this property="FormattedHtml" hintText="Field"/>
                </div>
              </swe: control>
            </li>
          </swe: for-each>
        </ul>
      </div>
      <div class="siebui-tile-image">
        <swe: for-each count="2" startValue="520" iteratorName="currentId">
          <swe: control id="swe: currentId" hintMapType="FormItem">
            <div class="siebui-form-data" align="swe: this.TextAlignment">
              <swe: this property="FormattedHtml" hintText="Field"/>
            </div>
          </swe: control>
        </swe: for-each>
      </div>
      <div class="siebui-tile-clear"/>
      <div class="siebui-tile-details-row2">
        <ul>
          <swe: for-each count="4" startValue="530" iteratorName="currentId">
            <li>
              <swe: control id="swe: currentId" hintMapType="FormItem">
                <div class="siebui-form-data" align="swe: this.TextAlignment">
                  <swe: this property="FormattedHtml" hintText="Field"/>
                </div>
              </swe: control>
            </li>
          </swe: for-each>
        </ul>
      </div>
    </swe: list-record>
  </swe: list>
</div>

```

```
</swe:List>
</div>
</div>
</swe:form>
<!-- Template End: CCAppl etLi st_Ti le. swt -->
```

Customizing Tiles for Mobile Lists

This topic describes how to customize tiles that the mobile tile renderer displays. A *mobile tile* is a type of user interface object that Siebel Open UI uses to display a Siebel CRM record as a small square or rectangle.

To customize tiles for mobile lists

- 1 Verify that your deployment includes the Applet Tile Mobile web template:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b In the Object Explorer, click Web Template.
- c In the Web Templates list, query the Name property for Applet Tile Mobile, and then verify that Siebel Tools includes an object definition for this web template.
- d In the Object Explorer, expand the Web Template tree, and then click Web Template File.
- e Verify that the Filename property references the CCAppl etLi st_Mobi le. swt file.

If your deployment does not include this web template, then add it now. Note that Siebel Open UI includes the predefined CCAppl etLi st_Mobi le. swt file in the `si ebel \webtempl \oui webtempl` folder, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4.

- 2 Modify the web template that the applet references:

- a In the Object Explorer, click Applet.
- b In the Applets list, query the Name property for the applet that you must modify.

For this example, query for the following predefined applet:

CG Contact Li st Appl et Mobi le

- c In the Object Explorer, expand the Applet tree, and then click Applet Web Template.
- d In the Applet Web Templates list, modify the following property of the Edit List applet web template.

| Property | Value |
|--------------|------------------|
| Web Template | Applet Tile Mode |

- e Right-click in the Applet Web Templates list, choose Edit Web Layout, and then use the Web Layout Editor to arrange columns, add columns, remove columns, and make other layout modifications, as necessary.

3 Compile your modifications.

4 Configure the manifest:

a Log in to the Siebel application with administrator privileges.

b Navigate to the Administration - Application screen, and then the Manifest Files list.

For more information about the screens you use in this step, see [“Configuring Manifests” on page 128](#).

c In the Files list, add the following file.

| Field | Value |
|-------|------------------------------|
| Name | siebel/TileLayoutMobilePR.js |

d Navigate to the Administration - Application screen, and then the Manifest Administration list.

e In the UI Objects list, add the following UI applet.

| Field | Value |
|------------|-------------------------------|
| Type | Applet |
| Usage Type | Physical Renderer |
| Name | CG Contact List Applet Mobile |

f In the Object Expression list, add the following expression.

| Field | Value |
|------------|--------|
| Expression | Mobile |
| Level | 1 |

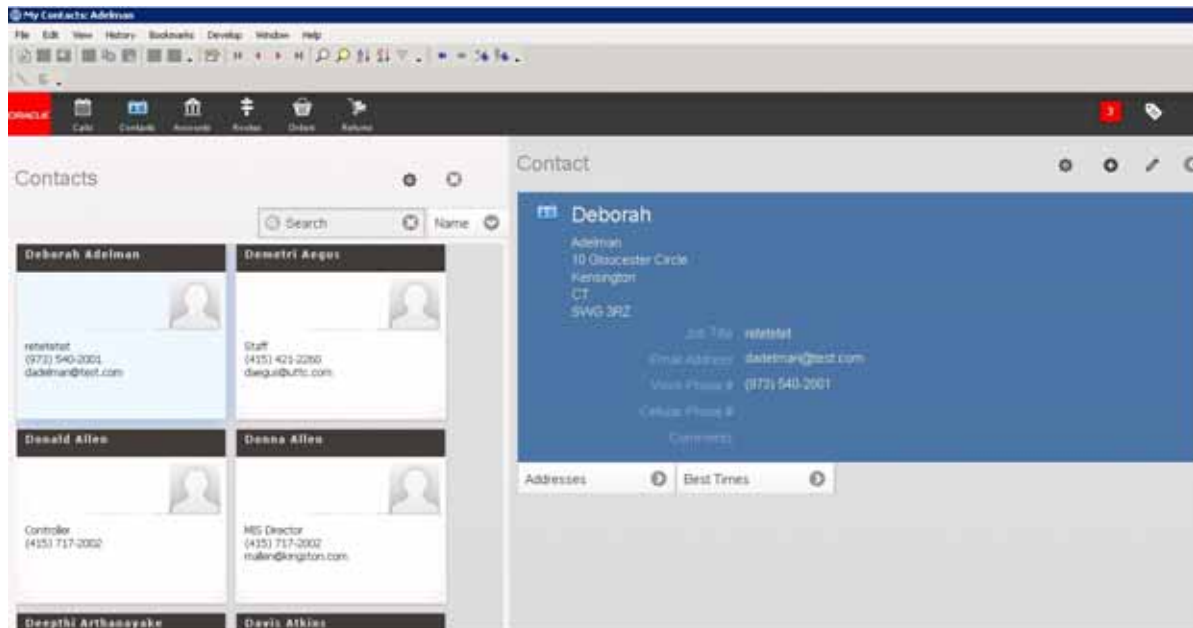
g In the Files list, add the following file.

| Field | Value |
|-------|--|
| Name | siebel/TileLayoutMobilePR.js |
| | Siebel Open UI uses the TileLayoutMobilePR.js file to render applet tiles, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4. It resides in the following folder: |
| | Si ebel \eapps\publ i c\enubui / d_number\scri pts\si ebel \ |

5 Test your modifications:

a Clear the cache, and then navigate to the applet that you specified in [Step e on page 313](#).

- b** Make sure Siebel Open UI displays an applet that is similar to the following, with the tiles organized on the left side of the applet and the form on the right.



Configuring Horizontal and Vertical Scrolling in Tile Applets

You can configure tiles so that the user can scroll them horizontally or vertically. The configuration you do to configure scrolling in tile applets is the same configuration that you do to configure infinite scrolling. For more information, see [“Configuring Infinite Scrolling” on page 340](#).

To configure horizontal and vertical scrolling in tile applets

- Add the following tags to the web template file that Siebel Open UI uses to display the tile applet. You can configure these tags when you edit the web template file in [Step 1 on page 306](#).

| SWE Tag | Description |
|---|--|
| swe: this property="NoGrid" | Specifies that the current list layout is not a Grid layout. |
| swe: this property="Horizontal" | <p>Specifies the scroll direction. You use the following format:</p> <pre><swe: this s property="direction" /></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>direction</i> is Horizontal or Vertical. <p>For example, the following code configures Siebel Open UI to display vertical scrolling:</p> <pre><swe: this s property="Vertical" /></pre> <p>If the web template file does not include this tag, or if this tag does not include a value then, then Siebel Open UI uses horizontal scrolling, by default. To view the layout that Siebel Open UI uses for these directions, see "Horizontal Scroll and Vertical Scroll Layouts" on page 315.</p> |
| swe: this property="ScrollVisibleTiles" | <p>Enables Scrolling By Visible Tiles. For more information, see "Scrolling By Visible Tiles" on page 316. If the web template file does not include this tag, or if this tag does not include a value, then Siebel Open UI uses Smooth Scrolling. For more information, see "Smooth Scrolling" on page 316.</p> |

Horizontal Scroll and Vertical Scroll Layouts

[Figure 37](#) includes an example of a horizontal scroll configured in a tile applet. The user can click the left or right arrow to scroll through the records.

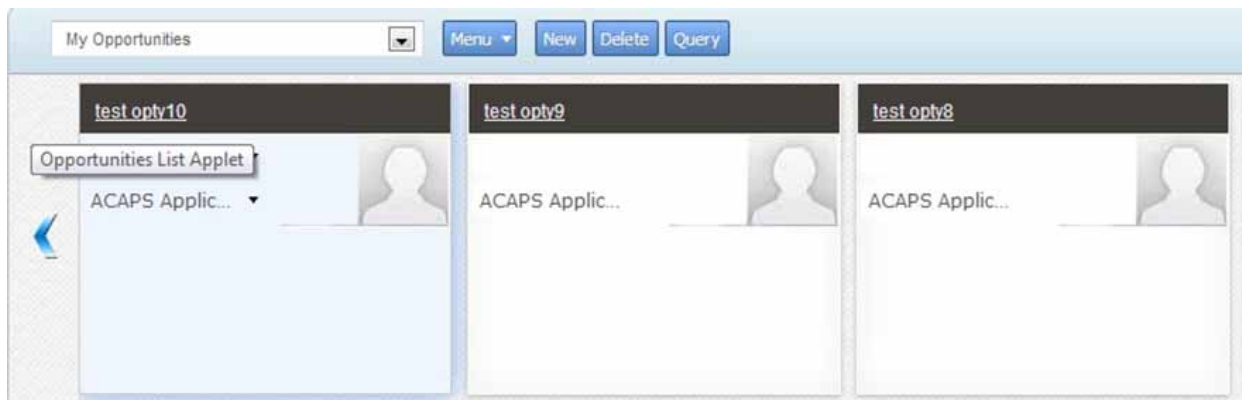


Figure 37. Example Horizontal Scroll in Tile Applet

Figure 38 includes an example of a vertical scroll configured in a tile applet. The user can click the up or down arrow to scroll through the records.

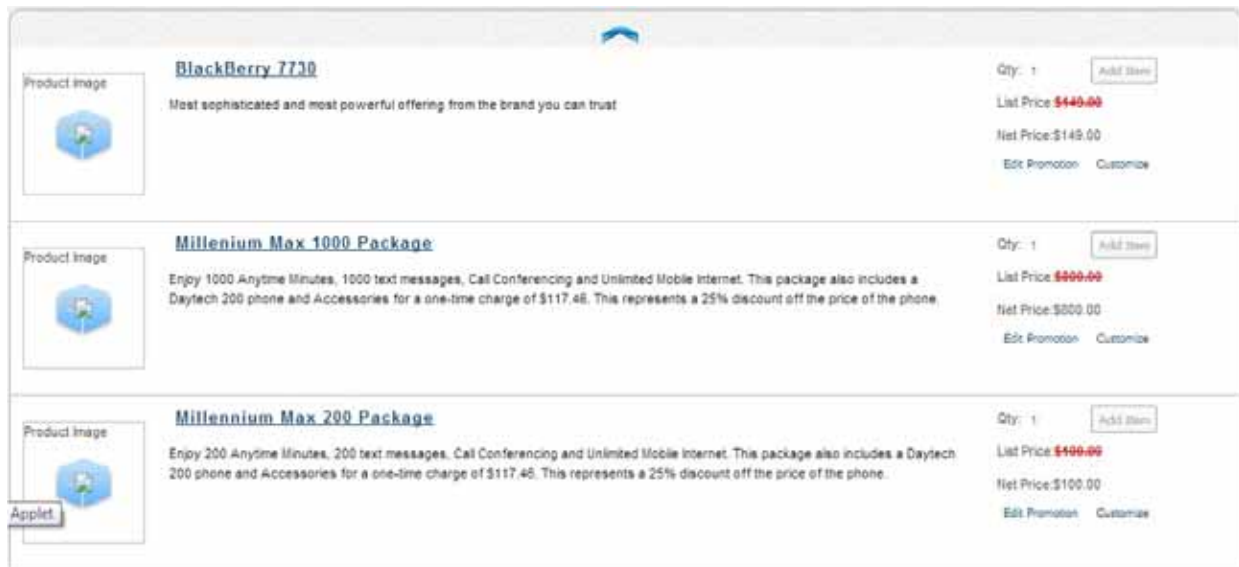


Figure 38. Example Vertical Scroll in Tile Applet

Smooth Scrolling

Smooth Scrolling is a feature that allows the user to hold down the mouse button on the scroll arrow to scroll the container according to the number of pixels that the scroll speed specifies. Siebel Open UI continues scrolling as long as the user holds down the mouse button. It stops scrolling when the user releases the mouse button. Siebel Open UI uses Smooth Scrolling, by default.

To set the scroll speed for all tile applets, the user can navigate to the Application screen, Tools, User Preferences, and then set the Default Scroll Speed field in the Behavior form. To set the scroll speed for a single tile applet, the user can use the slide control that each tile applet displays in the lower-right corner of the applet. For more information about how the user sets the scroll speed, see *Siebel Fundamentals*.

Scrolling By Visible Tiles

Scrolling By Visible Tiles is a feature that allows the user to click the mouse on the scroll arrow to start scrolling rather than holding down the mouse button on the scroll arrow. For each mouse click, Siebel Open UI scrolls the number of records according to the number of tiles that it displays in the tiles container rather than according to pixels.

For example, assume the record set includes 10 tiles and tile one through tile five of this record set are visible in the tile container. The width of the container determines the number of records that are visible. If the user clicks the right pointing scroll arrow, then Siebel Open UI scrolls the tiles container to display tiles six through ten.

Adding Controls to Tile Applets

Figure 37 illustrates how you can use a method to control the behavior of a button that Siebel Open UI displays in a tile applet. This example includes an Add Item button in a Product catalog that allows the user to add an item to the shopping cart. Each tile represents one product. Siebel Open UI uses the cascading style sheet to display the state of the Add Item button as enabled in the active tile and to display it as not chosen in other tiles. The CSS controls how Siebel Open UI represents these states. To add a control, you use Siebel Tools to add it to the applet, and then create a placeholder for it in the SWT file. For more information about adding controls to applets, see *Configuring Siebel Business Applications*.

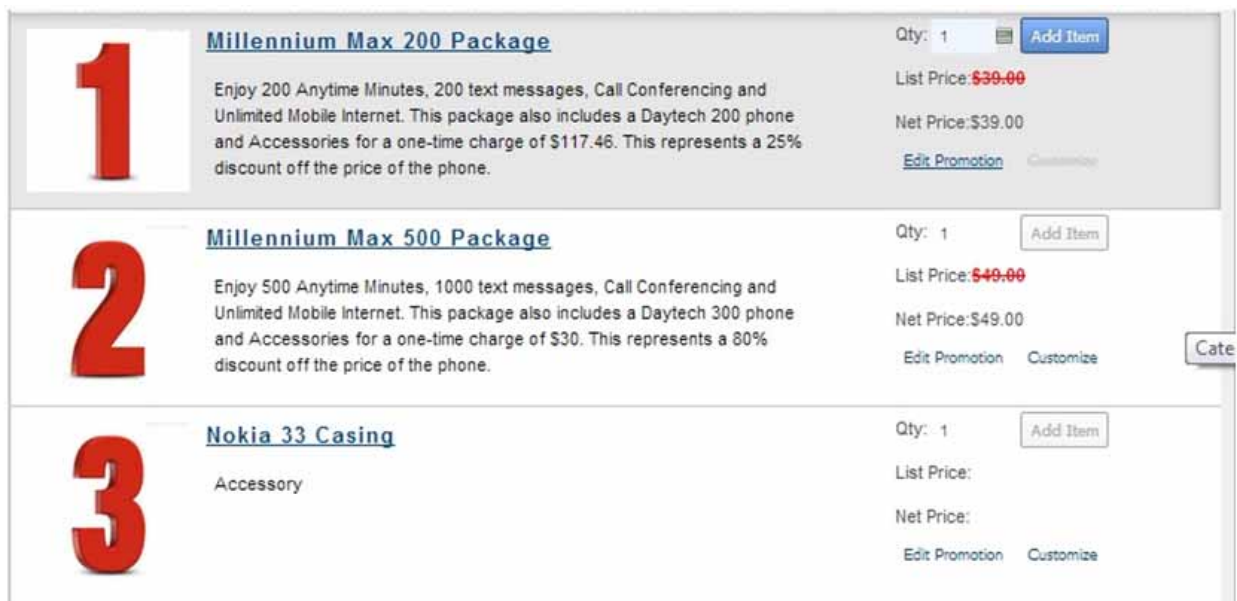


Figure 39. Example of Adding Controls to Tile Applets

Adding Toggle Controls

This topic describes how to add a toggle control that allows the user to toggle between displaying or hiding content in landscape or portrait layout.

To add toggle controls

- 1 Identify the .swt file that you must modify, and then open it for editing.

For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

2 Add a div element to the Web template file that you identified in [Step 1](#).

For more information, see [“Attributes That You Can Use with the Div Element for a Toggle Control” on page 318](#).

For example, the following code sets the display options for the back button that the `jqmToggleCtrl` control renders. It slides the element out of the viewport toward the right when the toggle displays the control, and it slides the element out of the viewport toward the left when the toggle hides the control:

```
<div name="jqmToggleCtrl" id="jqmToggleCtrl" ctrltype="button"
ctrllabel="IDS_SWE_MOBILE_BACK" effect="slide" showoptions='{direction:
'right'}' hideoptions='{direction: 'left'}' style="display: none;"></div>
```

For another example, the following code sets the display options for the back button that the `jqmToggleCtrl` control renders. It displays the element at 100% opacity when the toggle displays the control, and it displays the element at 0% opacity when the toggle hides the control:

```
<div name="jqmToggleCtrl" id="jqmToggleCtrl" ctrltype="button"
ctrllabel="IDS_SWE_MOBILE_BACK" effect="size" showoptions='{percent: '100'}'
hideoptions='{percent: '0'}' style="display: none;"></div>
```

Attributes That You Can Use with the Div Element for a Toggle Control

[Table 21](#) describes the attributes that you can use with the div element that determines how to display a toggle control.

Table 21. Attributes That You Can Use with the Div Element for a Toggle Control

| Attribute | Description |
|-----------|--|
| id | You set each of these attributes to the following value: jqmToggleCtrl Script in the client examines these attributes when it creates the control. |
| name | |
| ctrltype | You set this attribute to one of the following values: ■ button ■ link |
| ctrllabel | Sets the display label for the control. You set the value for this attribute to the key of the localized string value that the <code>swemessage_xxx.js</code> file contains. |
| style | Sets the default visibility for the control. You set this attribute to the following value: display: none; |

Table 21. Attributes That You Can Use with the Div Element for a Toggle Control

| Attribute | Description |
|-------------|--|
| effect | <p>Sets the animation effect that Siebel Open UI uses when the user clicks the toggle control. If you do not specify a value for this attribute, then Siebel Open UI uses no effect. You can use one of the following values:</p> <ul style="list-style-type: none"> ■ blind ■ clip ■ drop ■ explode ■ fold ■ puff ■ slide ■ scale ■ size ■ pulsate <p>For more information, see the page about user interface effects at the jQuery Web site at http://docs.jquery.com/UI/Effects/.</p> |
| showoptions | <p>Sets the display options for the effect. If you do not include showoptions, then Siebel Open UI sends empty options to the show method and uses the default that jQuery uses.</p> <p>You can use a pair of single quotation marks to enclose the value instead of the double quotation marks that JSON syntax requires. Siebel Open UI replaces these single quotation marks with double quotation marks so that showoptions complies with JSON. For more information, see the JSON Web site at http://www.json.org. For more information, see “Example Show and Hide Options” on page 319.</p> |
| hideoptions | <p>Behavior is the same as the behavior for the showoptions attribute, except hideoptions sets the hide options for the effect.</p> |

Example Show and Hide Options

The following code includes examples of the showoptions and hideoptions attributes:

```
effect="clip" showoptions="{ 'direction': 'horizontal' } hideoptions="{ 'direction': 'vertical' }"
effect="slide" showoptions="{ 'direction': 'left' } hideoptions="{ 'direction': 'right' }"
effect="scale" showoptions="{ 'percent': 100 } hideoptions="{ 'percent': 0 }"
effect="size" showoptions="{ 'to' : { 'width': 280, 'height': 185 } }"
hideoptions="{ 'to' : { 'width': 280, 'height': 185 } }"
```

Configuring Siebel Open UI to Toggle Row Visibility

This describes how to configure Siebel Open UI to toggle between displaying or hiding a row each time the user clicks this row.

To configure Siebel Open UI to toggle row visibility

- 1 Identify the .swt file that you must modify, and then open it for editing.

For example, CCAppletList_B_EL.swt. For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

- 2 Add the following div element to the Web template file that you identified in [Step 1](#):

```
div enabletoggle="true"
```

For example:

```
swe: switch>
  <swe: case condition="Web Engine State Properties, IsMobileApplicationMode">
    <div enabletoggle="true"></div>
  </swe: case>
</swe: switch>
```

You add this code to a different div element depending on whether or not you modify the web template that one of the following items use:

- **Applet.** Add the code to the root div element of the list applet.
- **View.** Add the code to the div element that is the parent of the root div element that the list applet uses.

If the user clicks a row, then Siebel Open UI calls the ToggleMobileLayout binder method only if it finds the child node that contains `div enabletoggle="true"`.

You can use the AttachPMBinding method to bind the ToggleMobileLayout method to an existing method. For more information, see [“AttachPMBinding Method” on page 428](#).

Siebel Open UI can bind the ToggleMobileLayout method to the HandleRowSelect method. The HandleRowSelect method detects row clicks on a list applet. The enabletoggle element enables this capability. For more information, see [“HandleRowSelect Method” on page 455](#).

Adding the Show More Button to Your Custom Form Applets

This topic describes how to configure Siebel Open UI to add the Show More Button to your custom form applets. This button allows the user to view controls that the applet contains, but that Siebel Open UI does not come predefined to display in the visible area of the screen.

To add the Show More Button to your custom form applets

- 1 Modify the Web template file:

- a Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b Choose the View menu, Windows, and then click Web Templates Window.
- c Scroll down in the Web Templates window, and then click the following file:

CCViewDetail_Mobile_RelatedItems

You modify a .swt file to add the Show More button. For this example, you modify the CCViewDetail_Mobile_RelatedItems.swt file that controls the contact form. For an example that includes details about how to identify and modify a web template, see ["Configuring Views to Use Landscape or Portrait Layout" on page 282](#).

Note that Siebel Open UI comes predefined to enable the Show More button for these .swt files. It uses these .swt files for all mobile form applets.

- d Right-click the window that displays the HTML code, and then choose Edit Template.
- e Locate the div element that Siebel Open UI uses to display the Show More button for the form applet container.

In this example, you modify the div element that contains the following code:

```
swe:applet hintMapType="Applet"
```

- f Add the following code to the div element that you located in [Step e](#):

```
<div id="_FormParentRelatedItems"
class="SiebelDetailParentContainerRelatedItems siebui-
NoScrollFormContainer">
  <swe:applet hintMapType="Applet" id="1" hintText="Parent Form Applet"
property="FormattedHtml" var="ParentWithPointer"/>
</div>
```

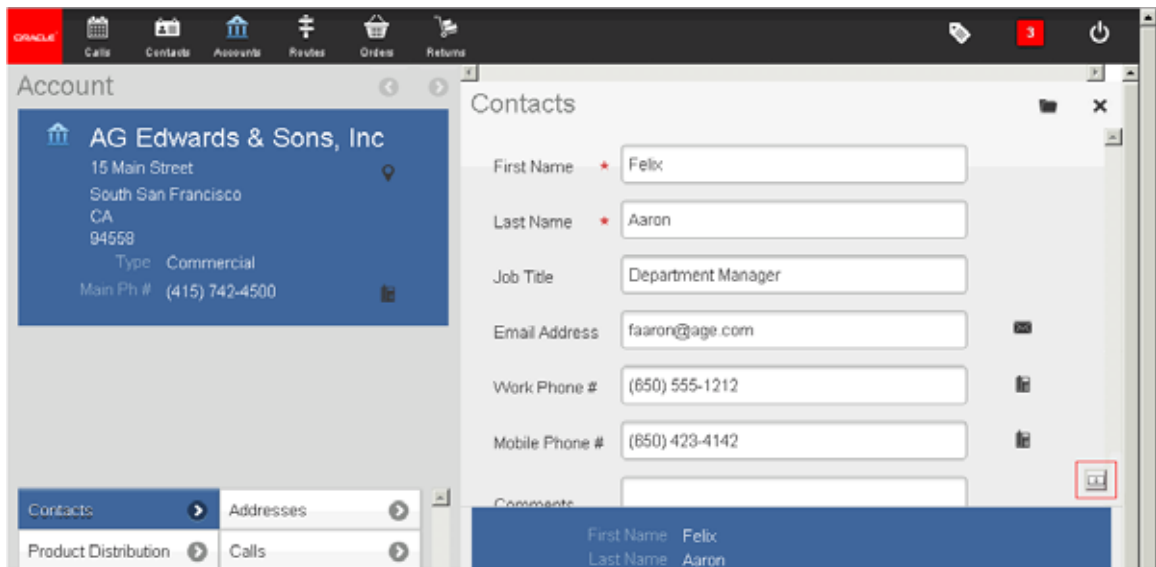
The siebui-NoScrollFormContainer class that resides in the .swt file controls the Show More feature. This class determines whether or not to display the Show More icon every time the user changes the orientation or resizes the window on the mobile device.

- 2 Test your modifications:

- a Use a mobile device to log in to the client, and then navigate to the Contact form.

- b** Verify that Siebel Open UI displays the Show More icon.

For example, Siebel Open UI displays the Show More icon to the right of the Comments field in the following image, as indicated by the red box.



- c** Click the Show More icon.
- d** Verify that Siebel Open UI expands the applet container, displays more fields, and displays the Show Less icon.
- Note that if all controls that the form applet contains fit into the visible area of the screen, then Siebel Open UI does not display the Show More icon.
- e** Click the Show Less icon.
- f** Verify that Siebel Open UI contracts the applet container, displays fewer fields, and displays the Show More icon.

Customizing Transitions, Themes, Styles, and Colors

This topic describes how to configure transitions, themes, styles, and colors. It includes the following information:

- [Customizing Transitions That Siebel Open UI Displays When It Changes Views on page 323](#)
- [Customizing Themes That Siebel Open UI Displays in Siebel Mobile Clients on page 325](#)
- [Customizing List Applet Styles on page 327](#)
- [Customizing jQuery Color Swatches That Siebel Open UI Displays in Siebel Mobile Clients on page 329](#)

Customizing Transitions That Siebel Open UI Displays When It Changes Views

The example in this topic describes how to add a custom CSS3 transition. A *CSS3 transition* is a visual effect that Siebel Open UI uses anytime it displays a view or refreshes a view after the user clicks New, Edit, Save, or Cancel in Siebel Mobile. Slide Left is an example of a transition. Siebel Open UI comes predefined with the Slide Left transition enabled. It automatically constrains the transition field depending on whether the user is using a desktop computer, tablet, or phone. The transition LOV field displays Slide Left and None for tablets and phones. If the user sets a transition on one platform, such as a tablet, and then uses another platform, such as a phone, then Siebel Open UI resets the transition to empty.

To customize transitions that Siebel Open UI displays when it changes views

- 1 Create a custom CSS file.
- 2 Use the following classes to add the transition to the file you created in [Step 1](#):

```
.siebui -prev-TransitionEffect-begin
.siebui -prev-TransitionEffect-end
.siebui -next-TransitionEffect-begin
.siebui -next-TransitionEffect-end
```

where:

- *TransitionEffect* identifies the transition effect name. For example, the following code specifies the slide-in transition effect:

```
.siebui -prev-slidein-begin {
position : absolute;
width : 100% !important;
top : 0;
z-index : 999;
-webkit-transition : top 1s ease-in-out;
-moz-transition : top 1s ease-in-out;
-o-transition : top 1s ease-in-out;
transition : top 1s ease-in-out;
}
.siebui -prev-slidein-end {
top : 2000px !important;
}
.siebui -next-slidein-begin {
margin-top : -2000px;
height : 0 !important;
-webkit-transition : margin-top 1s ease-in-out;
-moz-transition : margin-top 1s ease-in-out;
-o-transition : margin-top 1s ease-in-out;
transition : margin-top 1s ease-in-out;
}
.siebui -next-slidein-end {
margin-top : 0;
height : 100% !important;
}
```

3 Add the transition effect to the PAGE_TRANSITION list of values:

- a** Open Siebel Tools. Connect to the database that your Siebel Mobile application uses.
For more information, see *Using Siebel Tools*.
- b** Click the Screens application-level menu, click System Administration, and then click List of Values.
- c** Right-click in the List of Values list, click New Record, and then add the following value to the PAGE_TRANSITION list of values for tablets.

| Property | Value |
|---------------------------|-----------------|
| Type | PAGE_TRANSITION |
| Display Value | Slide In |
| Language-Independent Code | SLIDEIN |
| Parent LIC | Tablet |

- d** Right-click in the List of Values list, click New Record, and then add the following value to the PAGE_TRANSITION list of values for phones.

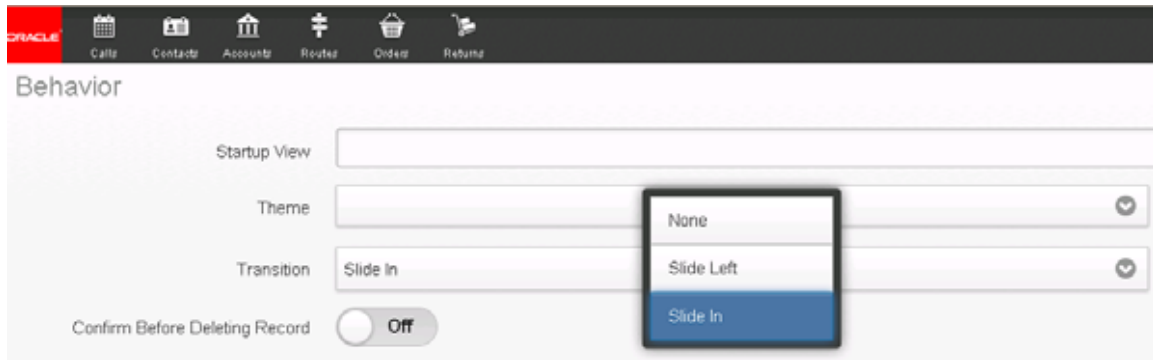
| Property | Value |
|---------------------------|-----------------|
| Type | PAGE_TRANSITION |
| Display Value | Slide In |
| Language-Independent Code | SLIDEIN |
| Parent LIC | Phone |

- e** Compile your modifications.

4 Test your modifications:

- a** Log in to the client.
- b** Navigate to the Setting screen.

- c Verify that Siebel Open UI displays the transitions you specified in [Step 2](#) in the Transition field. For example, verify that Siebel Mobile displays the Slide In transition.



- d Choose a value in the Transition field.
If you do not choose a transition or if you choose None, then Siebel Open UI does not use a transition.
- e Log out of the client, and then log back into the client.
- f Navigate between views, and verify that Siebel Open UI uses the transition you chose in [Step d](#).

Customizing Themes That Siebel Open UI Displays in Siebel Mobile Clients

The User Preferences - Behavior screen in the Siebel Mobile client allows the user to choose the theme that this client displays. Siebel Open UI comes predefined with one theme for the tablet and one theme for the phone, by default. It constrains the theme that the user can choose depending on whether the user uses a phone, tablet, or desktop computer. You can add a custom theme. The example in this topic describes how to add a custom theme named Mobile Theme Gold that Siebel Open UI displays on a tablet.

To customize themes that Siebel Open UI displays in Siebel Mobile clients

- 1 Create a new style sheet named theme-gold.css. Save this new file in the following folder:

`CLIENT_HOME\appsweb\PUBLIC\language_code\files\custom`

You can use any .css file that includes your custom theme. For this example, use theme-gold.css.

- 2 Use a JavaScript editor to open the tablettheme.js file that resides in the following folder:

`CLIENT_HOME\appsweb\PUBLIC\enu\release_number\scripts\siebel\custom`

If you must add a theme to a phone, then open the mobiletheme.js file instead of the tablettheme.js file.

- 3 Add the following custom theme to the file that you opened in [Step 2](#):

```
/* Adding a new custom theme */
siebel App.ThemeManager.addTheme(
    "theme_name",
    {
        CSS : {
            "css_id" : "files/custom/your_custom_file.css"
        }
    }
);
```

where:

- *theme_name* identifies the name of your custom theme.
- *your_custom_file* is a .css file that includes your custom color theme.

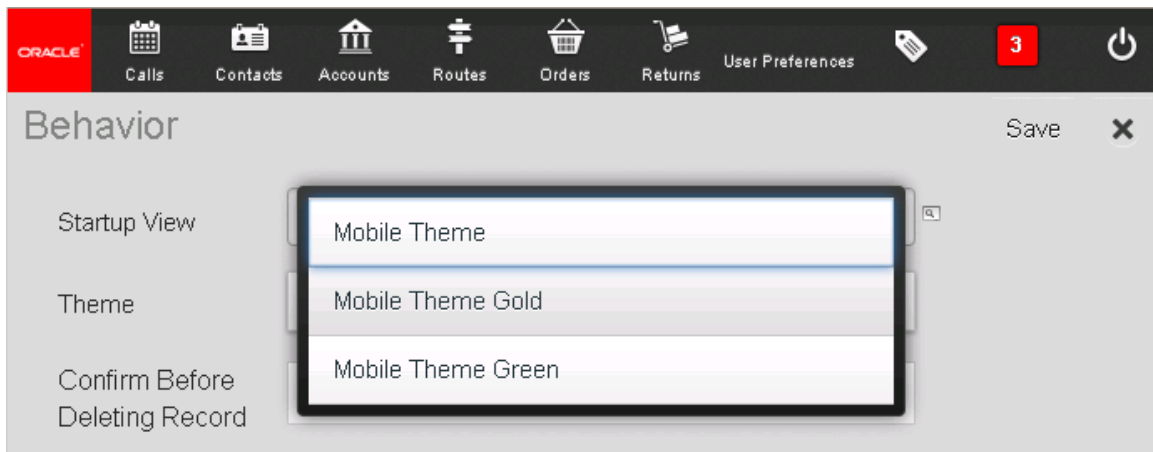
For this example, use the following code:

```
/* Adding a new custom theme */
siebel App.ThemeManager.addTheme(
    "MOBILE_THEME_GOLD",
    {
        CSS : {
            "css_id" : "files/custom/theme-gold.css"
        }
    }
);
```

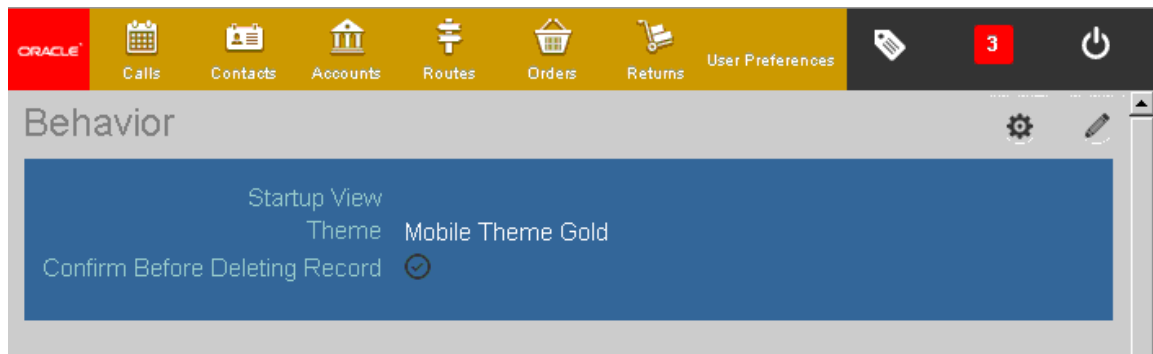
- 4 Add the new theme to the OUI_THEME_SELECTION list of values:
 - a Open Siebel Tools. Connect to the database that your Siebel Mobile application uses.
For more information, see *Using Siebel Tools*.
 - b Click the Screens application-level menu, click System Administration, and then click List of Values.
 - c Right-click in the List of Values list, and then click New Record.
 - d Add the following value to the OUI_THEME_SELECTION list of values.

| Property | Value |
|---------------------------|---|
| Type | OUI_THEME_SELECTION |
| Display Value | Mobile Theme Gold |
| Language-Independent Code | MOBILE_THEME_GOLD The value that you specify must match the theme name that you defined in Step 3 in the tablettheme.js file. In this example, this name is MOBILE_THEME_GOLD. |
| Parent LIC | NAVIGATION_BUTTON_TABLET If you are configuring for a phone, then use NAVIGATION_BUTTON_PHONE instead of NAVIGATION_BUTTON_TABLET. |

- 5 Test your modifications:
 - a Use a tablet to log in to the Siebel Mobile client.
 - b Click User Preferences, click Behavior, and then click Edit.
 - c Verify that the Theme field includes the following Mobile Theme Gold value:



- d Click Mobile Theme Gold, and then click Save.
 - e Log out of the Siebel Mobile client, and then log back in to this client.
 - f Verify that the Siebel Mobile client displays a theme that is similar to the following:



Customizing List Applet Styles

This topic describes how to customize the style that Siebel Open UI uses for each control that it displays in a list applet.

To customize list applet styles

- 1 Modify the applet:

- a** Open Siebel Tools.
For more information, see *Using Siebel Tools*.
- b** In the Object Explorer, click Applet.
- c** In the Applets list, query the Name property for the list applet that you must modify.
- d** In the Object Explorer, expand the Applet tree, expand the List tree, and then click List Column.
- e** In the List Columns list, query the Name property for the list column that you must modify.
- f** Specify the following property.

| Property | Description |
|----------------|---|
| HTML Attribute | <p>Enter the name of a class. For example, enter the following value:</p> <p style="text-align: center;"><code>Class = "account-location"</code></p> <p>You can enter any value that is unique within the HTML code that Siebel Open UI uses to render this list column. The value must be unique so that the client can correctly identify this attribute.</p> |

The HTML Attribute object type adds an HTML tag attribute to an HTML tag that the client creates when it displays the list column. In this example, the client creates an HTML tag attribute named account-location.

- g** Compile your modifications.
- 2** Specify the styling, and then test your modifications:
- a** Use a CSS editor to open the CSS file that Siebel Open UI uses to render the list applet.
For example, open the theme-mb-structure-ext.css file.
 - b** Specify the styling.

In this example, you add the following code to specify the color, font, justification, and so on, for the account-location class that you specified in [Step f](#):

```
.account-location{
  color: Olive;
  font-style: italic;
  float: right;
  width: 45%;
  text-align: right;
  margin: -2.2em 0;
  color: Fuchsia;
  font-weight: bold;
}
```
 - c** Save, and then close the CSS file.
 - d** Refresh your browser, and then make sure Siebel Open UI displays the styles that you specified.

Customizing jQuery Color Swatches That Siebel Open UI Displays in Siebel Mobile Clients

This topic describes how to use ThemeRoller to customize the color swatch that Siebel Open UI displays in Siebel Mobile clients. A *color swatch* is a jQuery Mobile user interface element that includes a header bar, content body, and button states. *ThemeRoller* is a jQuery Mobile tool that allows you to create color swatches.

Siebel Open UI uses the following items to define the style in a Siebel Mobile client:

- **Structure.** Defines web page layout.
- **Swatches.** Defines the color swatches in ThemeRoller format.
- **Style.** Provides style refinement capabilities.

Siebel Open UI uses the following file to define the structure of the JQM style, starting with Siebel CRM versions 8.1.1.11 and 8.2.2.4:

`jquery.mobile.structure-1.3.0.min.css`

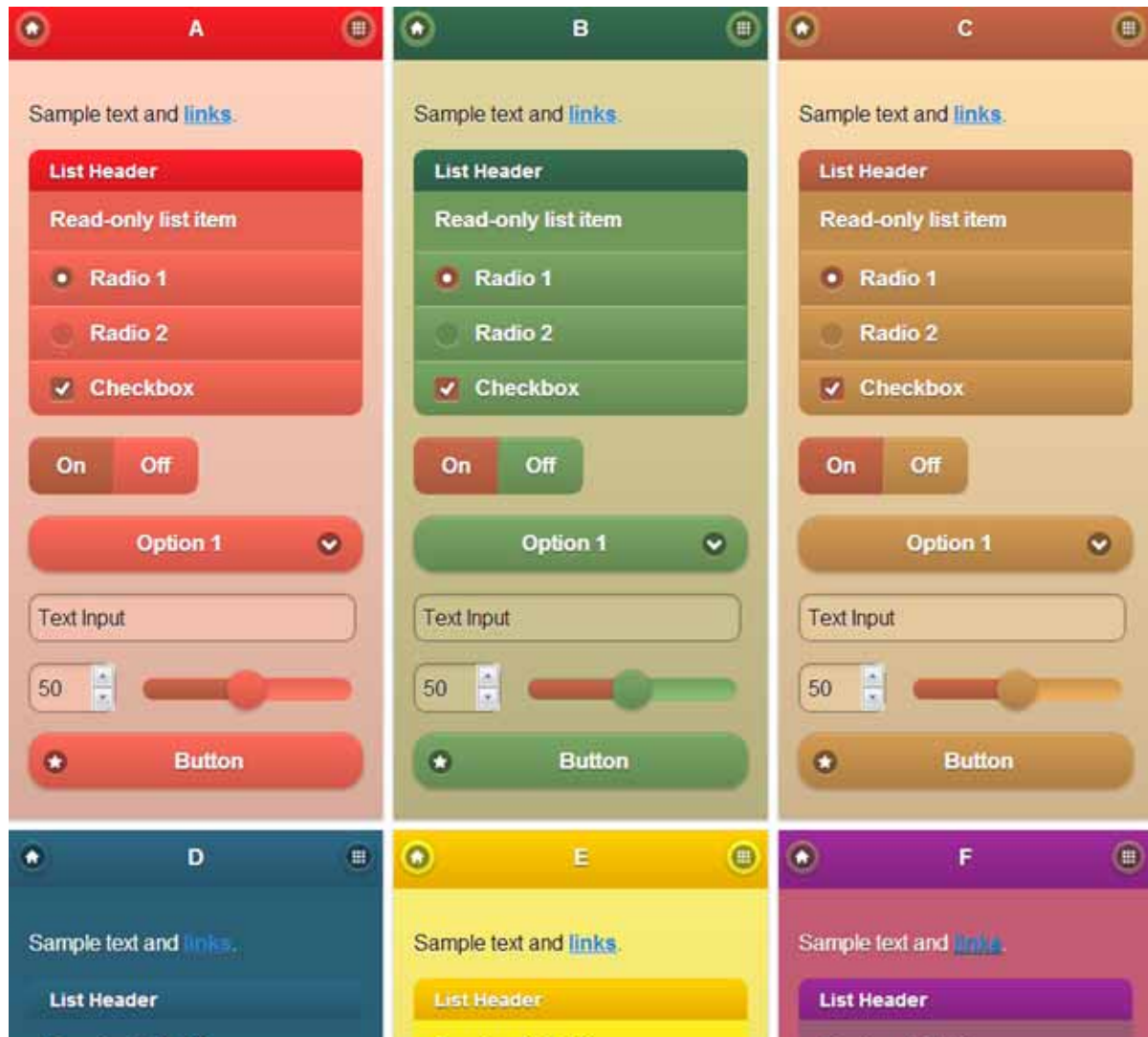
Siebel Open UI does not include the default JQM swatches. Instead, it uses the default Siebel Mobile swatches that the `theme-mb-swatches.min.css` file contains. These files replace the `jquery.mobile-1.1.1.min.css` file that Siebel Open UI uses in versions before Siebel CRM versions 8.1.1.11 and 8.2.2.4.

To customize color schemes that Siebel Open UI displays in Web pages

- 1 Open version 1.3.0 of the following jQuery Mobile ThemeRoller website:

<http://jquerymobile.com/themeroller/?ver=1.3.0>

- 2 Create six swatches lettered A through F, each of which uses a different color scheme.
For example:



For an alternative to creating these swatches, see [“Importing Predefined Siebel Open UI Swatches to ThemeRoller”](#) on page 332.

- 3 Click Download.
- 4 In the Download Theme dialog box, enter in a name in the Theme Name field, and then click Download Zip.

You can specify a name for the theme, such as my-swatches. Note that this name is different than the name of the zip file that you download. jQuery Mobile ThemeRoller automatically creates the name of the zip file and adds the following prefix to it:

jquery-mobile-theme-

- 5 Unzip the file that you downloaded in [Step 4](#).

- 6 Open the following folder of the file that you unzipped in [Step 5](#):

themes

- 7 Note the following items that reside in the themes folder:

- images subfolder
- my-swatches.css file
- my-swatches.min.css file

- 8 Copy the CSS files that you noted in [Step 7](#) to the folder that Siebel Open UI uses for cascading style sheets.

For example, copy these items to the following folder that resides in the Mobile Web Client client:

CLIENT_HOME\PUBLIC\language_code\files

For more information about the folders that you can use to store your customizations, see ["Organizing Files That You Customize" on page 122](#).

- 9 Copy the images that reside in the images folder to the image folder that your Siebel installation uses.

For example, the Mobile Web Client uses the following folder:

CLIENT_HOME\PUBLIC\language_code\images

- 10 Use an editor to open the my-swatches.min.css file.

- 11 Replace each occurrence of images with the following:

../images

- 12 Replace the theme-mb-swatches.min.css file with the my-swatches.min.css file:

- a Open the folder that Siebel Open UI uses for cascading style sheets, and then make a backup copy of the theme-mb-swatches.min.css file.
- b Rename the my-swatches.min.css file that you copied in [Step 6](#) to theme-mb-swatches.min.css.

If you prefer not to replace the theme-mb-swatches.min.css file, then do the following work:

- Navigate to the following folder, and then use a JavaScript editor to open the mobiletheme.js file:

ORACLE_HOME\siebsrvr\WEBMASTER\siebel_bui\ld\scripts\siebel\custom

- Add the following code:

```
Siebel App. ThemeManager. addResource(
    "SBL-MOBILE",
    {
        css : {
            "ss-theme" : "files/my-swatches.min.css"
```

```
    }  
  }  
);
```

- 13 Log in to the Siebel Open UI client, and then clear the browser cache.
- 14 Make sure Siebel Open UI displays the new color schemes.

Importing Predefined Siebel Open UI Swatches to ThemeRoller

This topic describes how to import the predefined swatches that Siebel Open UI uses to use ThemeRoller instead of creating them from scratch in [Step 2 on page 330](#).

To import predefined Siebel Open UI swatches to ThemeRoller

- 1 Use an editor to open the theme-mb-swatches.css file, and then copy the contents of this file to the clipboard.

The theme-mb-swatches.css file is the compressed version of the theme-mb-swatches.min.css file. It resides in the following folder:

`CLIENT_HOME\PUBLIC\language_code\files`

- 2 Use a web browser to navigate to the following website:
`http://jquerymobile.com/themeroller/?ver=1.3.0`
- 3 Click Import.
- 4 Paste the contents of the clipboard into the window in the Import Theme dialog box.
- 5 Click Import.
- 6 Modify the results, as necessary.

Swatches That Siebel Open UI Displays in Siebel Mobile Clients

[Table 22](#) describes the swatches that Siebel Open UI displays in Siebel Mobile clients.

Table 22. Swatches That Siebel Open UI Displays in Siebel Mobile Clients

| Swatch | Description |
|----------|---|
| Swatch A | Defines the application-level navigation bar. |
| Swatch B | Defines items that Siebel Open UI displays in a list. |
| Swatch C | Defines objects in list applets that some other swatch does not already define. |
| Swatch D | Defines the applet that Siebel Open UI displays in a grid. |

Table 22. Swatches That Siebel Open UI Displays in Siebel Mobile Clients

| Swatch | Description |
|----------|---|
| Swatch E | Defines the form applet that Siebel Open UI displays when the user navigates to the third level tab. |
| Swatch F | Defines the form applet that Siebel Open UI displays when the user navigates to the second level tab. |

Swatches That Siebel Open UI Displays in List Applets

Figure 40 includes an example of the swatches that Siebel Open UI displays in the Accounts list.

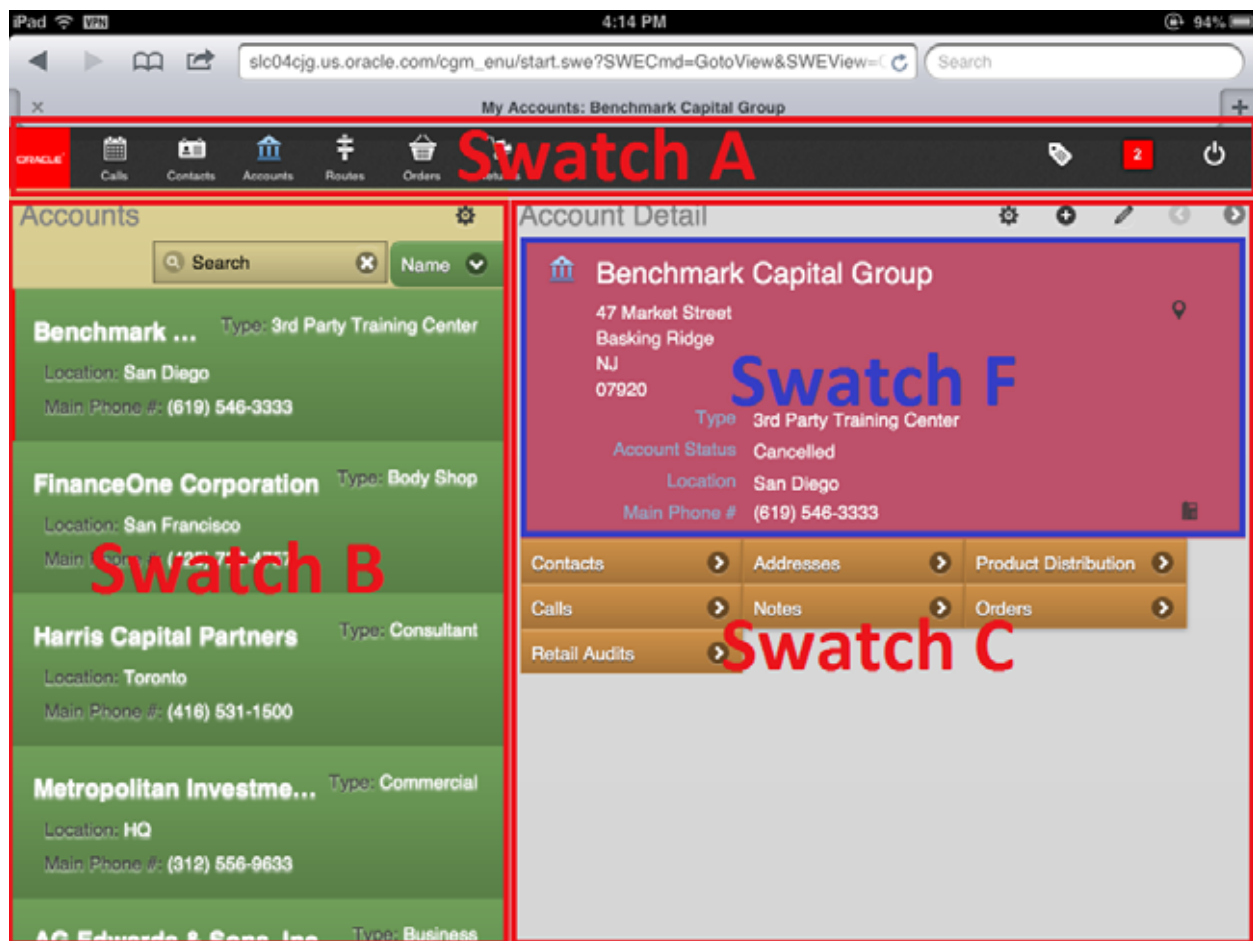


Figure 40. Swatches That Siebel Open UI Displays in List Applets

Swatches That Siebel Open UI Displays in Form Applets

Figure 41 includes an example of the swatches that Siebel Open UI displays in the Account form.

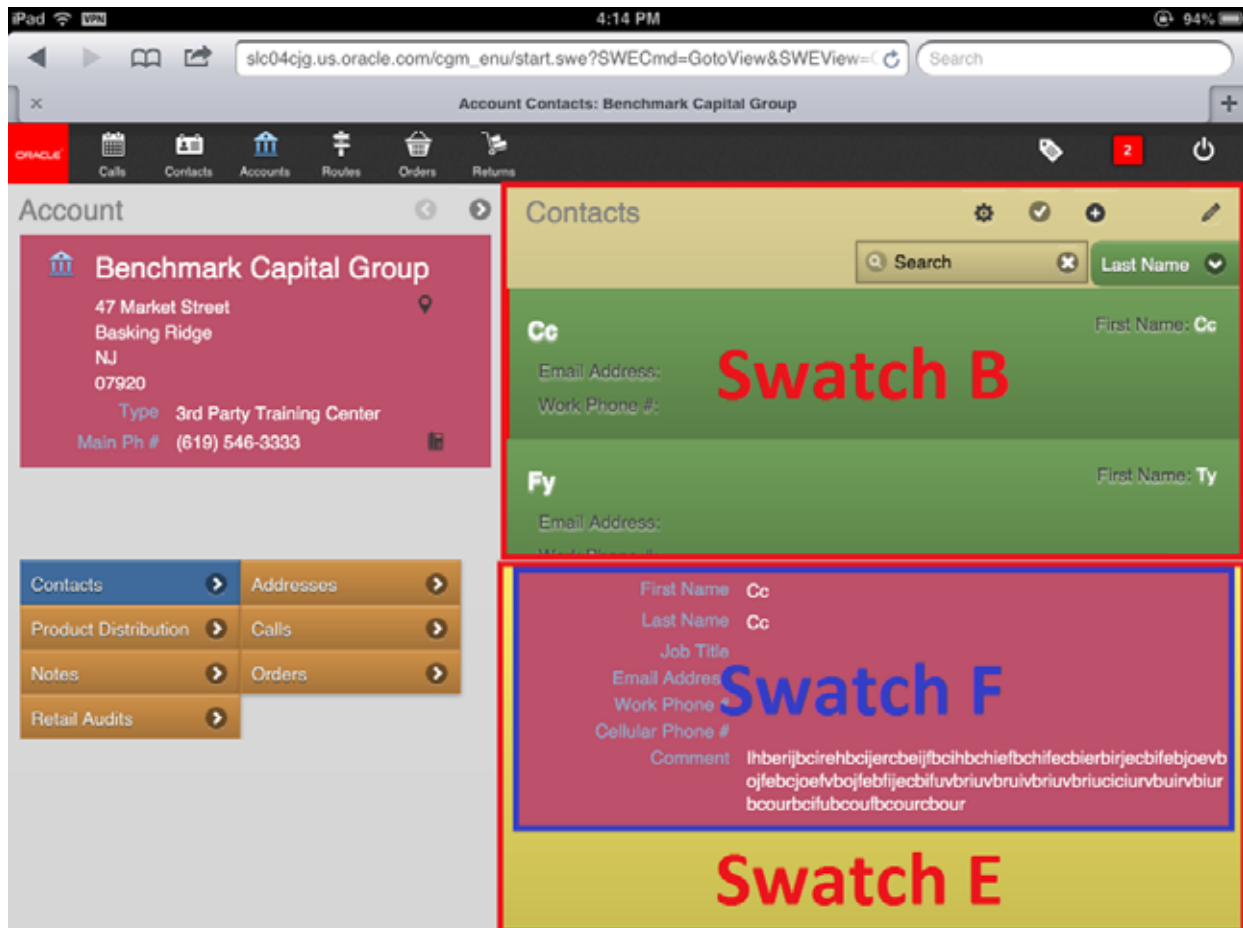


Figure 41. Swatches That Siebel Open UI Displays in Form Applets

Swatch That Siebel Open UI Displays in Grids

Figure 42 includes an example of the swatch that Siebel Open UI displays in a grid.

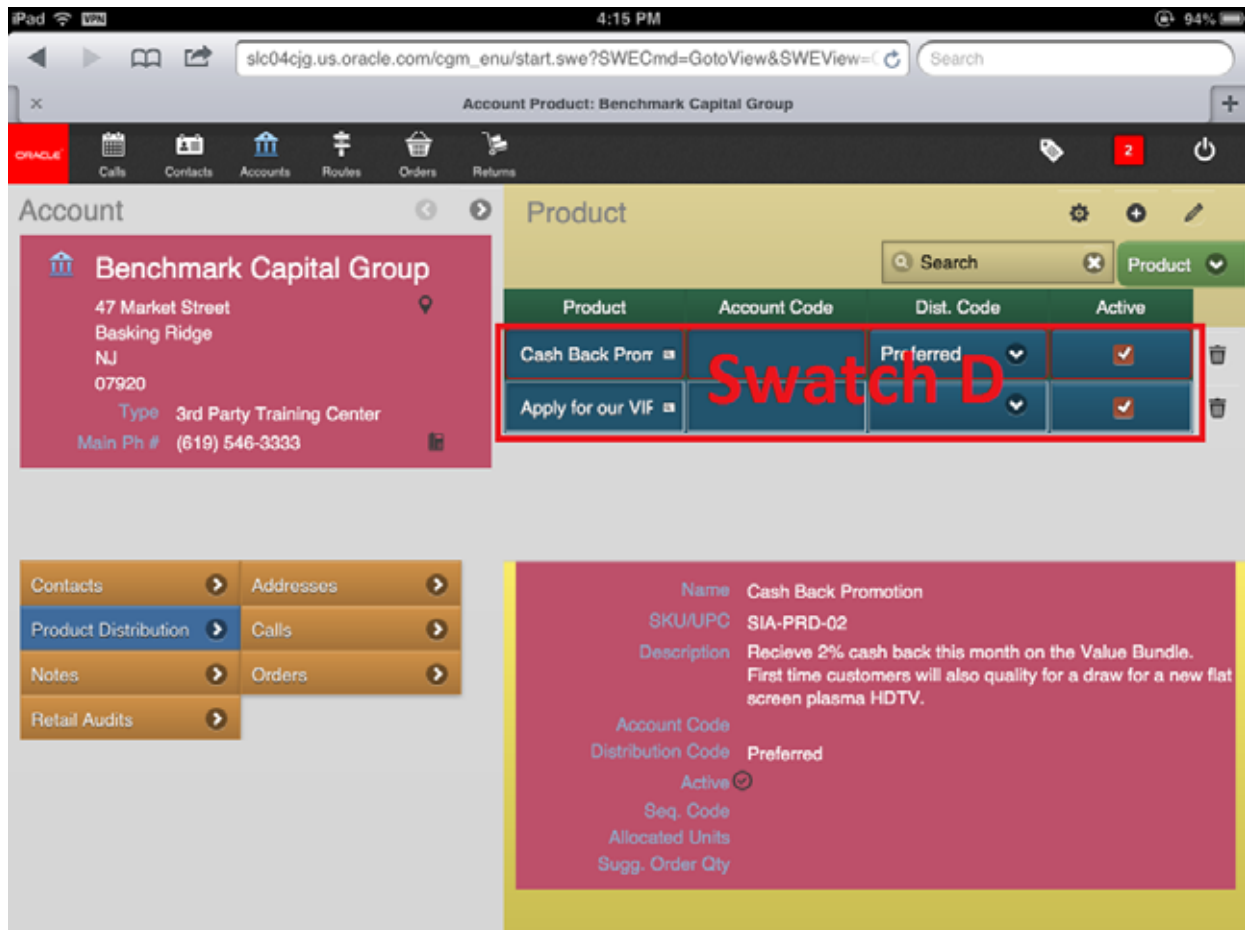


Figure 42. Swatch That Siebel Open UI Displays in Grids

Swatch That Siebel Open UI Displays in Pop-Up Objects

Figure 43 includes an example of the swatch that Siebel Open UI displays in a pop-up object.

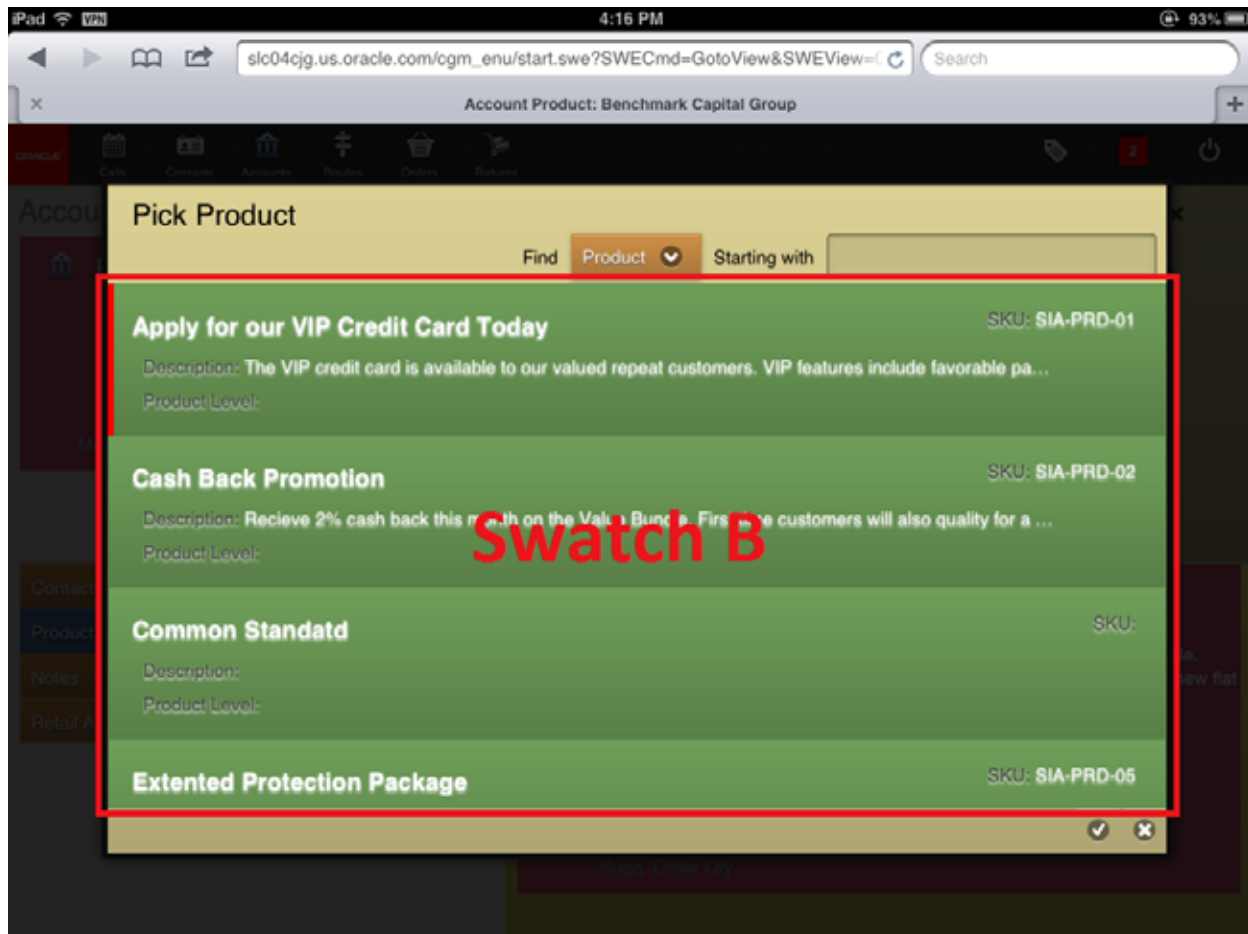


Figure 43. Swatch That Siebel Open UI Displays in Pop-Up Objects

Customizing Scrolling and Swipe

This topic describes how to configure scrolling and swipe. It includes the following information:

- [Configuring Generic Scrolling in Siebel Mobile on page 337](#)
- [Configuring Swipe Scrolling on page 339](#)
- [Configuring Infinite Scrolling on page 340](#)
- [Configuring the Height of the Scroll Area on page 341](#)
- [Configuring Swipe to Delete on page 342](#)

For information about configuring scrolling in a tile applet, see [“Configuring Horizontal and Vertical Scrolling in Tile Applets” on page 314](#).

Configuring Generic Scrolling in Siebel Mobile

Siebel Open UI uses the following physical renderers to render a list applet in Siebel Mobile:

- jqmrenderer
- jqmgridrenderer

Siebel Open UI uses the `jqmscrollcontainer` predefined class for each of these renderers or for any other renderer that renders scrolling in a list applet.

The `jqmscrollcontainer` supports pagination only in the Y direction. It does not support pagination in the X direction. It supports only one scroll container for one physical renderer for each list applet. A *scroll container* is an object that contains an HTML div element. Siebel Open UI makes any HTML markup that this div element contains scrollable.

To configure generic scroll in Siebel Mobile

- 1 Identify the method you must use to configure generic scrolling.

For more information, see [“Methods You Can Use to Configure Generic Scrolling” on page 338](#).

- 2 Edit the method that you identified in [Step 1](#).

Methods You Can Use to Configure Generic Scrolling

Table 23 describes the methods that you can use with the `jqmscrollcontainer` class to configure generic scrolling. You can configure the physical renderers that use this class to directly reference these methods.

Table 23. Methods You Can Use to Configure Generic Scrolling

| Method | Description |
|-------------------------|---|
| EnableListScrolling | <p>You use the following syntax:</p> <pre>EnableListScrolling = function(container_ID, list, x_direction, y_direction)</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>container_ID</i> identifies the ID of the scroll container. This container is the parent container where this method creates a child div. Siebel Open UI makes any HTML markup that this child div contains scrollable. ■ <i>list</i> contains one of the following values: <ul style="list-style-type: none"> ■ true. The physical renderer uses the <code>jqmlistrenderer</code> to render the data in a list. ■ false. ■ <i>X_direction</i> contains one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI scrolls in the X direction. ■ false. ■ <i>Y_direction</i> contains one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI scrolls in the Y direction. ■ false. |
| MakecontainerScrollable | <p>You use the following syntax:</p> <pre>MakecontainerScrollable = function(container_ID, list, x_direction, y_direction)</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>container_ID</i> identifies the ID of the scroll container. This method creates this div and uses it as the scroll container. <p>The <code>MakecontainerScrollable</code> method uses all other arguments in the same way that the <code>EnableListScrolling</code> method uses them.</p> |

Table 23. Methods You Can Use to Configure Generic Scrolling

| Method | Description |
|--------------------|--|
| SetScrollcontainer | <p>You use the following syntax:</p> <pre>SetScrollcontainer = function(Scrollviewcontainer)</pre> <p>Scrollviewcontainer: is same as above.</p> <p>If you configure Siebel Open UI to call this method, then you must make sure this method sends this argument as a null value. Siebel Open UI sets this method to the scroll container in the predefined class.</p> |
| GetScrollcontainer | <p>You use the following syntax:</p> <pre>GetScrollcontainer = function()</pre> <p>This method gets the div Id of the scroll container that Siebel Open UI sets in the predefined class.</p> |

Configuring Swipe Scrolling

The desktop Siebel Open UI client uses next and previous buttons to allow the user to scroll through a record set. Siebel Mobile uses automatic pagination on swipe scrolling instead of next and previous. For example, if a page can display 10 records, then it might display fewer records to meet display requirements, and allow the user to can scroll up or down to view more records. If the user encounters the 10th record and keeps scrolling, then Siebel Open UI sends a request to the Siebel Server for more records, displays these new records in the client removes the old records. To configure this behavior, you can edit the following files:

- jquery.easing.1.3.js
- jquery.mobile.scrollview.js
- jquery.mobile.scrollview.css

If the user stops the scroll in the up direction or in the down direction, then Siebel Open UI starts the Touchend event. If the user scrolled out of the current page, then it sends a request to the Siebel Server for the next or for the prior set of records.

Configuring Infinite Scrolling

You can customize how Siebel Open UI displays *infinite scrolling*, which is a feature that allows the user to scroll through records in a list applet indefinitely. Siebel Open UI continues to scroll records as long as the user clicks the key that controls scrolling.

To configure *infinite scrolling*

- 1 Set the following parameter in the `InfraUIFramework` section of the application configuration file:

```
ScrollAmountSize = integer
```

You can set the `ScrollAmountSize` parameter to an integer value. The default value is 5. For more information, see [“Modifying the Application Configuration File” on page 105](#).

- 2 Save your modifications, and then close the application configuration file.
- 3 Log in to the Siebel Server, and then enter the following Server Manager command:

```
change param ScrollAmountSize=value for comp applicationObjMgr_enu
```

where:

- *value* is an integer
- *application* identifies a Siebel application

For example:

```
change param ScrollAmountSize=50 for comp ePharmamObjMgr_enu
```

- 4 Set the following object manager parameters for the object manager that your Siebel application uses.

| Parameter | Description |
|------------------|--|
| NumberOfListRows | Specify the number of rows that the Siebel application displays in list applets. The default value is 10. To specify the number of rows for an individual applet, you can use the HTML Number of Rows property of the applet. For more information about this property, see <i>Siebel Object Types Reference</i> . |
| ScrollAmountSize | Specify the number of records that Siebel Open UI scrolls each time the user clicks the key that controls scrolling. For example, if you enter the following value, then Siebel Open UI scrolls 15 records: 15 The default value is 5. |

To make sure Siebel Open UI scrolls smoothly, it is recommended that you set NumberOfListRows to a value that is three times greater than the value you set for ScrollAmountSize.

For example:

NumberOfListRows = 21

ScrollAmountSize = 7

For another example:

NumberOfListRows = 30

ScrollAmountSize = 10

For more information about setting object manager parameters, see *Siebel System Administration Guide*.

Configuring the Height of the Scroll Area

Siebel Open UI sets the height of the scroll area that it displays in a list applet to the height of the view area of the device, by default. You can configure this height.

To configure the height of the scroll area

- 1 Identify the .swt file that the list applet that you must modify references, and then open this .swt file for editing.

For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

- 2 Add the ListAppletContainer class to the container div.

Configuring Swipe to Delete

You can configure Siebel Open UI so that if the user swipes a record in a list applet from left to right, then it displays a Delete button that allows the user to confirm the deletion. The user can click Delete to delete the record, or step off the record or tap it to cancel the deletion. Swipe to Delete comes enabled with Siebel Open UI, by default.

To configure *Swipe to Delete*

- 1 Identify the list applet that you must modify:
 - a Log in to the client.
 - b Navigate to the list applet where you must configure Swipe to Delete.
For example, navigate to the Accounts list.
 - c Click the application-level Help menu, and then click About View.
 - d In the About View dialog box, notice the name of the list applet.
For example, CG Account List Applet.
 - e Click OK.
- 2 Modify the DeleteRecord control:
 - a Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b In the Object Explorer, click Applet.
 - c In the Applets list, query the Name property for the list applet you noticed in [Step 1](#).
 - d In the Object Explorer, expand the Applet tree, and then click Control.
 - e In the Controls list, query the Name property for DeleteRecord, and then modify the following property.

| Property | Value |
|----------------|--------------|
| Method Invoked | DeleteRecord |

If the DeleteRecord control does not exist, then add it now. Note that Swipe to Delete only displays the Delete button if the user swipes from left to right on a record. It does not display a permanent Delete button. For more information, see [“DeleteRecord Method” on page 395](#).

 - f Compile your modifications.
- 3 Test your modifications:
 - a Log in to the client, and then navigate to the applet you noticed in [Step 1](#).
 - b Swipe a record from left to right.
 - c Make sure Siebel Open UI displays the Delete button.

- d** Step off the record and make sure Siebel Open UI hides the Delete button.
- e** Swipe a record from left to right, and then click Delete.
- f** Make sure Siebel Open UI deletes the record.

To disable Swipe to Delete, remove the control that you modified in [Step 2](#), and then compile your modifications.

How Siebel Open UI Uses Swipe to Delete

Siebel Open UI uses the CanDelete method in the jquery.swipeButton.min.js file that resides in the following folder to control Swipe to Delete:

```
CLIENT_HOME\eappweb\PUBLIC\language_code\release_number\scripts\3rdParty\jquerymobile
```

The CanDelete method makes sure that Siebel Open UI places the focus on the list item and makes sure that the Delete button is enabled or disabled according to the CanInvoke reply that the DeleteRecord method sends. If the applet that you query in [Step c on page 342](#) does not include a control with a Method Invoked property set to DeleteRecord, then Siebel Open UI does not display a button when the user swipes a record from left to right.

Customizing How Siebel Open UI Interacts with Siebel Mobile Applications

- [Adding Maps That Include Location Data in Siebel Mobile on page 343](#)
- [Configuring Siebel Open UI to Display Siebel CRM Data on Google Maps on page 346](#)

Adding Maps That Include Location Data in Siebel Mobile

This topic describes how to configure Siebel Open UI to add maps that include location data in Siebel Mobile. Siebel Open UI uses the location of the mobile device service to map contacts. It can access these services through methods. The location service allows you to use the zip code or postal code to map contacts, accounts, or activities. For more information, [“Method That Integrates Google Maps” on page 514](#).

To add maps that include location data in Siebel Mobile

- 1** Make sure you prepare the data that this example requires.
For more information, see [“Setting Up Configuration for Siebel Mobile Examples” on page 277](#).
- 2** Add a map button to the Contact List Applet:

- a** Open Siebel Tools. Choose the Siebel Mobile tag in the log in dialog box.
For more information, see *Using Siebel Tools*.
- b** In the Object Explorer, click Applet.
- c** In the Applets list, query the Name property for Contact List Applet.
- d** In the Object Explorer, expand the Applet tree, expand the List tree, and then click List Column.
- e** In the List Columns list, add the following list column.

| Property | Value |
|---------------------------------|----------------------------|
| Name | Personal Full Address |
| Field | Personal Full Address |
| Display Name - String Reference | SBL_ADDRESS-1004224839-ONU |

- f** In the Object Explorer, click Control.
- g** In the Controls list, add the following control.

| Property | Value |
|----------------------------|------------------------|
| Name | ShowAccounts |
| Caption - String Reference | SBL_MAP-1009093849-828 |
| HTML Type | MiniButton |
| Method Invoked | ShowMapLocations |

- 3** Configure Siebel Open UI to display the Address field:
 - a** In the Object Explorer, click Applet Web Template.
 - b** In the Applet Web Templates list, query the Name property for Edit List.
 - c** In the Object Explorer, expand the Applet Web Template tree, and then click Applet Web Template Item.
 - d** In the Applet Web Template Items list, query the Name property for Account, and then set the Inactive property to TRUE.
 - e** In the Applet Web Templates list, right-click the Edit List record, and then click Edit Web Layout.
 - f** In the Web Layout Editor, drag and drop the following list column from the Controls/Columns Palette to an appropriate location in the canvas:
`' Personal Full Address' (Display Name: Address)`
 - g** Drag and drop the following control from the Controls/Columns Palette to an appropriate location in the canvas:
`' ShowAccounts' (Display Name: Map)`
 - h** Save your modifications, and then close the Web Layout Editor.

- i** In the Object Explorer, click Applet.
 - j** In the Applets list, right-click the Contact List Applet, and then click Compile Selected Objects. Compile this applet into the SRF that the client uses.
- 4** Modify the mobile contact list view to use the correct web template:
- a** In the Object Explorer, click Web Template.
 - b** In the Web Templates list, add the following web template.

| Property | Value |
|----------|------------------------|
| Name | View Detail Map Mobile |
| Project | Mobile LS |
| Type | View Template |

- c** In the Object Explorer, expand the Web Template tree, and then click Web Template File.
- d** In the Web Template Files list, add the following web template file.

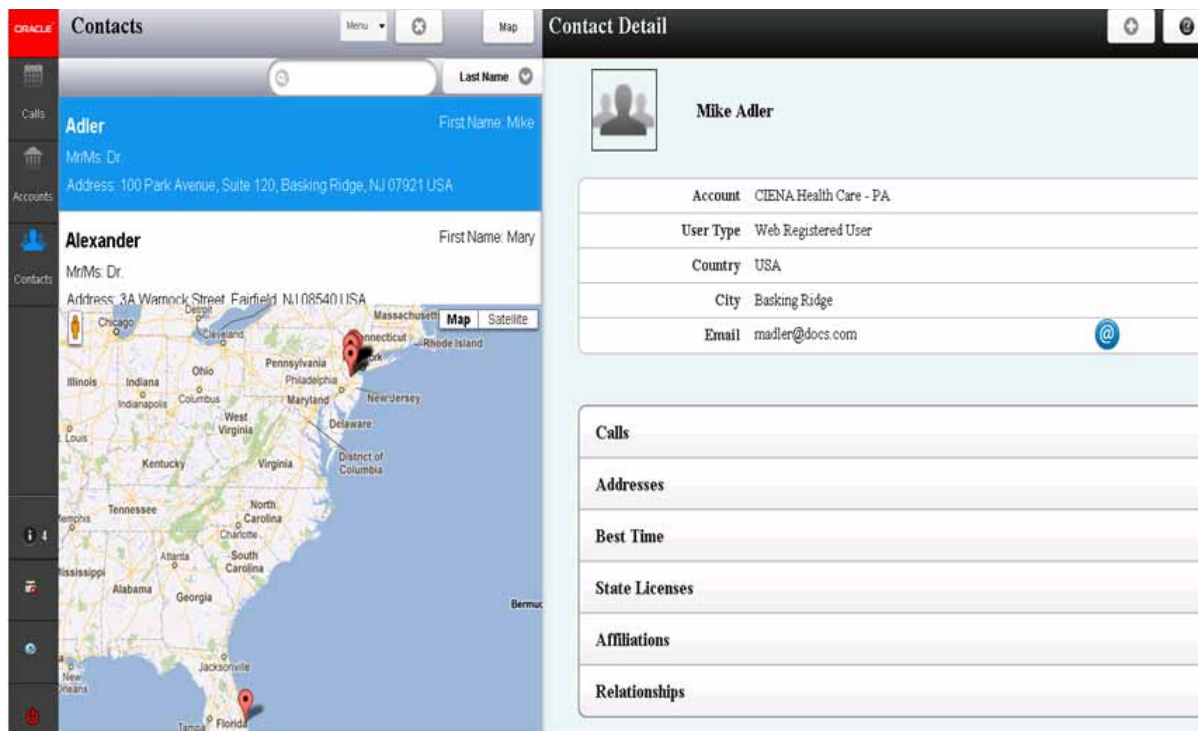
| Property | Value |
|-----------|---------------------------------|
| Name | View Detail Map Mobile Template |
| File Name | CCViewDetailMap_Mobile.swt |
| Bitmap | CCViewDetailMap_Mobile.swt |

- e** Right-click the View Detail Map Mobile Template record, and then click Compile Selected Objects. Compile this file into the SRF that the client uses.
- f** In the Object Explorer, click View.
- g** In the View list, query the Name property for Pharma Contact List View - Mobile.
- h** In the Object Explorer, expand the View tree, and then click View Web Template.
- i** In the View Web Templates list, query the Name property for Base, and then set the following property.

| Property | Value |
|--------------|------------------------|
| Web Template | View Detail Map Mobile |

- j** In the Views list, right-click the Pharma Contact List View - Mobile record, and then click Compile Selected Objects. Compile this file into the SRF that the client uses.
- 5** Test your modifications:
- a** Use Safari to open the ePharmacy application.
 - b** Reset Safari for mobile mode.
 - c** Navigate to the Contacts screen.

- d** Make sure Siebel Open UI displays a Map button on the contact list applet.
 - If Safari does not display the map, then make sure you set the Safari browser proxy correctly.
- e** Click the Map button and make sure Siebel Open UI displays a Google map at the bottom of the list applet.
- f** Make sure the map includes markers that identify the physical location of the contacts. For example:



Configuring Siebel Open UI to Display Siebel CRM Data on Google Maps

In this topic you configure Siebel Open UI to display Siebel CRM data in a Google map. If the user clicks a map button, then Siebel Open UI displays a Google map that can include markers that identify the current location of the mobile user, markers that identify the locations of various Siebel CRM address records, and routing information. It allows the user to get driving directions and to view other information, such as the customer name and account revenue for each marker.

Displaying Siebel CRM Data in a Google Map in a Separate Window

In this topic you configure Siebel Open UI to display Siebel CRM data in a Google map in a separate window. It displays this window if the user clicks a map icon from a field in a form applet. It references a field that includes the location information that it displays in the Google map.

To display Siebel CRM data in a Google map in a separate window

1 Modify the applet:

- a** Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b** In the Object Explorer, click Applet.

- c** In the Applets list, query the Name property for the form applet that you must modify.

For example, query the Name property for Account Form Applet.

- d** In the Object Explorer, expand the Applets tree, and then click Control.

- e** In the Controls list, query the Name property for the control that you must modify.

For example, query the Name property for the BillToStreetAddress control.

- f** Note the name of the value in the Field property.

Siebel Open UI will display the map icon in this field. For example, the Street Address field.

- g** In the Object Explorer, click Applet User Prop.

- h** In the Applet User Properties list, add the following applet user property.

| Property | Value |
|----------|--|
| Name | MAP_LOCATION_FIELDS |
| Value | <p>Enter the name of a business component field. For example:</p> <p> Billing Address</p> <p>To specify more than one field, use a comma to separate each field name. For example:</p> <p> Billing Address, Shipping Address</p> <p>At run-time, Siebel Open UI will create a separate push pin in the Google map for each field that you specify.</p> |

- i** Compile your modifications.

2 Test your modifications:

- a** Log in to the client.

- b** Verify that Siebel Open UI displays a map icon in the field that you noted in [Step f](#).

For example, navigate to the Accounts screen, click Accounts List, and then verify that Siebel Open UI displays a map icon in the City field in the form applet.

- c** Click this map icon, make sure Siebel Open UI opens a new window, and then displays a Google map in this window.

- d Make sure this map includes route and direction information for the route that exists between the current geographic location of the mobile user and the address location of the current Siebel CRM address record.

Displaying Siebel CRM Data in a Google Map in a List Applet

In this topic you configure Siebel Open UI to display Siebel CRM data in a Google map in a list applet. If the user clicks a map button in a field in a list applet, then Siebel Open UI displays this map immediately below the list of records in this list applet.

To display Siebel CRM data in a Google map in a list applet

- 1 Add the map button:
 - a Open Siebel Tools.
For more information, see *Using Siebel Tools*.
 - b In the Object Explorer, click Applet.
 - c In the Applets list, query the Name property for the list applet that must contain the map button.
 - d In the Object Explorer, expand the Applet tree, and then click Control.
 - e In the Controls list, add the following control.

| Property | Value |
|----------------|------------------|
| Name | Enter any value. |
| Caption | Enter any value. |
| HTML Type | MiniButton |
| Method Invoked | ShowMapLocations |

The ShowMapLocations method dynamically loads the Google map method, initializes the geocoder service to get the latitude and longitude of the address, and creates a marker for each location that exists in the array.

Siebel Open UI can call this method from a list applet. To call the method, it creates a button or link control and binds a click event with the control. In `jqmListRenderert`, it loops through the set of records in the list applet, examines the available columns and the non-null value of address fields of the record. To create the full address and add it in the array, it binds the ShowMap button control that is defined in the web template with the click event that the `jqmListRenderer` defines and calls the ShowMapLocations method in the `JQMMMapCtrl` class.

- f In the Object Explorer, click Applet Web Template.
- g In the Applet Web Template list, query the Name property for Edit List.
- h In the Object Explorer, expand the Applet Web Template tree, and then click Applet Web Template Item.

- i In the Applet Web Template Items list, add the following applet web template item.

| Property | Value |
|----------|---|
| Name | Enter any value. |
| Control | Choose the control that you added in Step e . |

- 2 Add an applet user property:

Do [Step h on page 347](#) except do this step for the button that you added in [Step 1 on page 348](#).

If this applet user property references a field that Siebel Open UI does not display in the client, then do the following work:

- a In the Object Explorer, click Applet.
- b In the Applets list, locate the applet that you must modify.
- c In the Applets list, click the link that Siebel Tools displays in the Business Component property.
- d In the Object Explorer, expand the Business Component tree, and then click Business Component User Prop.
- e In the Business Component User Properties list, add the following business component user property.

| Property | Value |
|----------|---|
| Name | PrivateFields |
| Value | Enter the name of a business component field. For example: B i l l i n g A d d r e s s To specify more than one field, use a comma to separate each field name. For example: B i l l i n g A d d r e s s , S h i p p i n g A d d r e s s |

- 3 Modify the .swt file:

- a Identify the .swt file that you must modify, and then open it for editing.

For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

- b Add the following code immediately after the SiebelContentListContainer div element:

```
<swe:if condition="Web Engine State Properties, IsMobileApplicationMode">  
  <div id="Siebel MapContainer" name="Siebel MapContainer"  
    style="display: none;">
```

```
<div id="jqmMapCtrl" name="jqmMapCtrl" ></div>
</div>
</swe:if>
```

This code adds the map container and div element. Siebel Open UI uses it to render the inline map.

- c** Optional. To specify a current location that Siebel Open UI uses if the current geolocation is not available, you can add the following code:

```
<div id="jqmMapCtrl" name="jqmMapCtrl" myLocation="address"></div>
```

For example:

```
<div id="jqmMapCtrl" name="jqmMapCtrl" myLocation="8 New England Executive
Park, Burlington, MA 01803 USA"></div>
```

Removing Map Buttons and Icons from Applets

This topic describes how to remove the map button from a list applet and the map icon from a form applet.

To remove map buttons and icons from applets

- 1** Modify the applet:

- a** Open Siebel Tools.

For more information, see *Using Siebel Tools*.

- b** In the Object Explorer, click Applet.
- c** In the Applets list, locate the applet that you must modify.
- d** In the Object Explorer, expand the Applet tree, and then click Control.
- e** In the Controls list, query the Method Invoked property for the following value:
ShowMapLocations
- f** Set the Inactive property to Y for all records that Siebel Tools returns.
- g** In the Object Explorer, click Applet User Prop.
- h** In the Applet User Properties list, query the Name property for MAP_LOCATION_FIELDS.
- i** Set the Inactive property to Y for all records that Siebel Tools returns.

- 2** Optional. If the Value property of a MAP_LOCATION_FIELDS applet user property references a field that Siebel Open UI does not display in the client, and if no other applet uses this field for the location service then, then you can remove this field from the business component user property. Do the following:

- a** In the Object Explorer, click Applet.
- b** In the Applets list, click the link that Siebel Tools displays in the Business Component property.

- c In the Object Explorer, expand the Business Component tree, and then click Business Component User Prop.
 - d In the Business Component User Properties list, search the Name property for the following value:

PrivateFields
 - e Set the Inactive property to Y for all records that Siebel Tools returns.
 - f Set Inactive = Y (if value is a single field).
 - g Remove the field from the list if the value includes more than one field.
- 3 Compile your modifications.
- 4 Modify the Web template file:
- a Identify the .swt file that you must modify, and then open it for editing.
 - b Remove the following code:

```
<swe:if condition="Web Engine State Properties, IsMobileApplicationMode">  
  <div id="SiebelMapContainer" name="SiebelMapContainer"  
    style="display: none;">  
    <div id="jqmMapCtrl" name="jqmMapCtrl" ></div>  
  </div>  
</swe:if>
```

For a similar example that identifies and modifies a web template, see [“Configuring Views to Use Landscape or Portrait Layout” on page 282](#).

Removing Map Buttons and Icons from All Applets

This topic describes how to remove the map button from all list applets and the map icon from all form applets.

To remove map buttons and icons from some applets

- Location service auto check the availability of the Google method. It disables Map button on the list applets and removes Map icon from the form applet.

10 Customizing Siebel Open UI for Siebel Mobile Disconnected

This chapter describes how to customize Siebel Open UI for Siebel Mobile disconnected. It includes the following topics:

- [Overview of Customizing Siebel Open UI for Siebel Mobile Disconnected on page 353](#)
- [Doing General Customization Tasks for Siebel Mobile Disconnected on page 356](#)
- [Customizing Siebel Pharma for Siebel Mobile Disconnected Clients on page 372](#)
- [Customizing Siebel Service for Siebel Mobile Disconnected Clients on page 375](#)
- [Methods You Can Use to Customize Siebel Mobile Disconnected on page 387](#)

Overview of Customizing Siebel Open UI for Siebel Mobile Disconnected

This topic describes an overview of customizing Siebel Open UI for Siebel Mobile disconnected. It includes the following information:

- [“Operations You Can Customize When Clients Are Offline” on page 353](#)
- [“Operations You Cannot Customize When Clients Are Offline” on page 354](#)
- [“Process of Customizing Siebel Open UI for Siebel Mobile Disconnected” on page 355](#)

For examples that customize Siebel Open UI for Siebel Mobile and for Siebel Mobile disconnected, see [Chapter 9, “Customizing Siebel Open UI for Siebel Mobile.”](#)

Operations You Can Customize When Clients Are Offline

You can customize the following operations when the client is offline:

- Create, read, update, and delete parent objects and child objects.
- Modify user interface behavior according to data characteristics, such as read only, required, and can invoke. Siebel Open UI uses the `IsReadOnly`, `IsRequired`, and `CanInvoke` methods to achieve this behavior.

You can customize the following items when the client is offline:

- Association applets
- Applet menu and applet menu items
- Pick applets
- Picklists

- Static picklists
- Error statuses
- Static drill downs
- Expressions
- Searches

Operations You Cannot Customize When Clients Are Offline

You cannot customize the following operations when the client is offline:

- Multivalue fields.
- Multivalue groups.
- Dynamic controls. A *dynamic control* is a type of control that Siebel Open UI creates dynamically at run time. The Siebel repository does not specify a dynamic control. For example, a view might contain a placeholder for a control that Siebel Open UI dynamically creates and displays at run time.
- Dynamic drilldowns.
- Toggle applets.
- Language-dependent code conversion to language-independent code. The Siebel Server does this conversion during synchronization.
- Custom layout modification.
- Effective dating. The Siebel EAI Adapter allows Siebel Open UI to access effective dating data. *Effective dating data* is data that identifies the start date and the end date for a field or link. A third-party application can request and receive effective dating data from the Siebel application. For more information about effective dating, see *Overview: Siebel Enterprise Application Integration* and *Siebel Public Sector Guide*.
- Siebel Application Response Measurement (SARM) usage.
- Siebel eScript or Siebel Visual Basic usage. Scripts that reside on the Siebel Server do not work in an offline client, so you must migrate them to JavaScript that resides on the client. Business service scripts do work in offline clients.
- Drilldown visibility. Siebel Open UI comes predefined to use the visibility that the drill down definition specifies. If this definition does not exist, or if it contains no values, then Siebel Open UI uses the view to determine drilldown behavior. If the view does not specify drilldown behavior, then Siebel Open UI uses business component visibility in the following order to determine drilldown behavior:
 - SalesRep
 - Personal
 - Org

- Numeric totals in applets. Some applets display the total for a series of numbers that reside in a column in a list applet or for all records. Siebel Open UI cannot display these totals while the client is offline.
- COM object usage, such as runtime events, data maps, or variable maps.
- Cascade delete.
- Search specification on a link.
- Sort specification that includes a date field.
- User properties for various objects except for the user properties associated with items described in [“Operations You Can Customize When Clients Are Offline” on page 353](#).
- Default applet menu items.
- Workflow processes.
- CreateRecord method.
- New record creation from an association popup applet. Siebel Open UI comes predefined to disable this creation. You can customize Siebel Open UI to enable it.

Note the following offline behaviors:

- Siebel Open UI displays only the data that it downloads during a full download for any business component field that it populates through a join that joins different tables.
- If more than one business component references the same table, and if Siebel Open UI modifies a business component record for one of these business components, then it does not populate this modification to the other business components until the user goes online and synchronizes the client with the Siebel Server.
- If the Owner Delete property of a business component is set to TRUE, then the user cannot delete a record in this business component even if this user owns or creates this record. This user must go online to delete the record. For more information about this property, see *Siebel Object Types Reference*.

Process of Customizing Siebel Open UI for Siebel Mobile Disconnected

It is recommended that you use the sequence of steps that this topic describes to customize Siebel Open UI to use a Siebel application in a Disconnected client. Siebel Pharma and Siebel Service are each an example of a Siebel application. To view examples that use these steps, see [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#) and [“Customizing Siebel Service for Siebel Mobile Disconnected Clients” on page 375](#).

To customize Siebel Open UI for Siebel Mobile Disconnected

- 1 Configure the manifest, if necessary.

For more information, see [“Modifying Manifest Files for Siebel Mobile Disconnected” on page 356](#).

2 Create a new JavaScript file or copy an existing one.

You must place all custom presentation models and physical renderers in a custom folder. For more information about this folder, see [“Organizing Files That You Customize” on page 122](#).

3 Register your custom JavaScript method or Siebel business service.

For more information, see [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#).

4 Add your custom code:

- a** Declare your variables.
- b** Use the `CanInvokeMethod` method to make sure Siebel Open UI can call your custom method or business service.
- c** Specify the logic for your custom JavaScript method or Siebel business service.
- d** Use `InvokeMethod` to call your custom JavaScript method or Siebel business service.

For more information, see [“Using Custom JavaScript Methods” on page 364](#).

5 Test your modifications.

Doing General Customization Tasks for Siebel Mobile Disconnected

This topic describes how to do general customization tasks for Siebel Mobile disconnected in Siebel Open UI. It includes the following topics:

- [“Modifying Manifest Files for Siebel Mobile Disconnected” on page 356](#)
- [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#)
- [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#)
- [“Using Custom JavaScript Methods” on page 364](#)
- [“Using Custom Siebel Business Services” on page 367](#)
- [“Configuring Data Filters” on page 369](#)
- [“Configuring Objects That Siebel Open UI Does Not Display in Clients” on page 369](#)
- [“Configuring Error Messages for Disconnected Clients” on page 369](#)

Modifying Manifest Files for Siebel Mobile Disconnected

The cache manifest file specifies the resources that Siebel Open UI must download to the disconnected client for offline use. Each application uses a separate cache manifest file that uses the following format:

application_name.manifest

where:

- *application_name* identifies the name of the Siebel application, such as Siebel Service for Mobile. Siebel Open UI converts this name to lower case and replaces each space that the name contains with an underscore. For example, siebel_service_for_mobile.manifest is the cache manifest file that Siebel Open UI uses for Siebel Service for Siebel Mobile disconnected.

Manifest files reside in the following folder on the Mobile Web Client:

```
\SWEApp\PUBLIC\language_code\siebel_service_for_mobile.manifest
```

Manifest files reside in the following folder on the Siebel Server:

```
\SES\siebsrvr\WEBMASTER\language_code\siebel_service_for_mobile.manifest
```

Siebel Open UI includes only the cache manifest files that it requires to support the Siebel application that you deploy.

To modify manifest files for Siebel Mobile disconnected

- 1 Add resources to the cache manifest file that your application uses, as necessary.

If your deployment requires custom resources to run an application offline, then you must add these resources to the cache manifest file that this application uses. For example, assume you must configure Siebel Open UI to run Siebel Service for Siebel Mobile disconnected so that it can download the following resources, and then use them while the client is offline:

- my_style.css
- my_image.png
- my_script.js

In this situation, you can create a file named my_cache.manifest that includes the following information:

```
CACHE MANIFEST
# 2012-4-27: v1
# Explicitly cached 'master' entries.
CACHE:
files/my_style.css
images/my_image.png
23012/scripts/my_script.js
```

The cache manifest file must use the HTML 5 standard. This standard allows you to run a Perl script in [Step 4](#) that merges your custom cache manifest files into the predefined application cache manifest files. Siebel Open UI includes this script starting with the Siebel CRM 8.1.1.10 Quick Fix release.

- 2 Make a backup copy of the predefined manifest file that you must modify.

For example, siebel_service_for_mobile.manifest. You modify this file in [Step 4](#).

It is recommended that you do this backup because the script that you run in this task modifies the siebel_service_for_mobile.manifest file. You can use this backup if you encounter a problem when running this script.

- 3 Open a Windows command line on the computer where the manifest files reside, and then navigate to the following folder:

```
\SWEApp\PUBLIC\language_code\mergemanifest.pl
```

The SWEApp folder resides on the Mobile Web Client. If you are doing this task on the Siebel Server, then navigate to the following folder:

```
\SES\siebsrvr\WEBMASTER\language_code\mergemanifest.pl
```

- 4 Enter the following command:

```
Perl mergemantest.pl -s my_cache.manifest -d application_name.manifest
```

where:

- *my_cache.manifest* specifies the source manifest file. If you do not include the -s switch, then Siebel Open UI uses the custom.manifest file, by default.
- *application_name.manifest* specifies the destination manifest file. You must include the -d switch.

For example:

```
Perl mergemantest.pl -s my_cache.manifest -d siebel_service_for_mobile.manifest
```

This command merges the custom manifest file that you modified in [Step 1](#) into the predefined siebel_service_for_mobile.manifest file. Note the following:

- You must run this script any time you modify your cache manifest file or do an upgrade.
- You must make sure the source and destination files exist.
- This script adds the CACHE, NETWORK, and FALLBACK sections that reside in the *my_cache.manifest*, if they exist, to the end of the corresponding sections that reside in the siebel_service_for_mobile.manifest file. Your custom entries take precedence over the predefined Oracle entries that reside in this file.
- If a file contains more than one CACHE section, NETWORK section, or FALLBACK section, then this script merges these sections into one section. For example, if two CACHE sections exist, then this script merges these CACHE sections into a single CACHE section. This merge does not modify the sequence where the entries reside in the files.
- The script does not add duplicate entries to the destination file. If the merge results in duplicate entries, then Siebel Open UI removes the first duplicate from the destination file. It adds this removed entry to the *destination.log* file that resides in the folder where the destination file resides.
- The script does not include empty lines in the destination file.
- This script creates the *destination.log* file every time it runs.
- If the script finishes the merge, and if the result of this merge is identical to the destination file, then the script does not update the destination file, and the destination file retains its original timestamp.

Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence

Siebel Mobile disconnected uses a *local database*, which is a database that resides in the browser that stores the data that Siebel Open UI requires. If Siebel Open UI runs in a Siebel Mobile disconnected environment, and if a method in JavaScript code interacts with this local database, then this method is an *asynchronous method*, and any method that calls this method is also an asynchronous method. This situation might result in Siebel Open UI running code in an incorrect sequence. For example, it might run section B of the code before it runs section A, but the correct logic is to run section A, and then run section B. This topic describes how to configure Siebel Open UI to call an asynchronous method as if the call occurred in a synchronous environment. This configuration makes sure Siebel Open UI runs the asynchronous method in the correct sequence.

To register methods to make sure Siebel Open UI runs them in the correct sequence

- 1 On the client computer, use a JavaScript editor to open the file that includes the business service call that you must modify.

For more information, see [“Using Custom JavaScript Methods” on page 364](#).

- 2 Locate the code that includes the business service call that you must modify.
- 3 If the call to any asynchronous method from a business service method must run only one time, then use the callback method and the setReturnValue method.

For example, you can use the ExecuteQuery and FirstRecord methods. Assume you locate the following code in [Step 2](#):

```
business_service.prototype.Submit = function () {
    retObj = bc.ExecuteQuery();
    err = retObj.err;
    if(!err){
        retObj = bc.FirstRecord();
        if(!retObj.err){
            //Do an operation here that sets the return value to bRet
            return({err: false, retVal: bRet});
        }
    }
    else{
        SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("messageKey",
errParamArray);
        return({err: true});
    }
};
```

- where *business_service* identifies the name of the business service that your custom code calls. For example, PharmaCallSubmitsvc.

For more information, see [“SetErrorMsg Method” on page 420](#), [“FirstRecord Method” on page 396](#) and [“ExecuteQuery Method” on page 395](#).

In this example, you replace the code that you located in [Step 2](#) with the following code:

```
PharmaCallSubmitsvc.prototype.Submit = function () {
    bc.ExecuteQuery();
    $.callback(this, function(retObj){
        err = retObj.err;
        if(!err){
            bc.FirstRecord();
            $.callback(this, function(retObj){
                if(!retObj.err){
                    //Do an operation here that sets the return value to bRet
                    $.setReturnValue({err: false, retVal: bRet});
                }
            });
        }
        else{
            SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("messageKey",
errParamArray);
            $.setReturnValue({err: true});
        }
    });
};
```

The callback method and the setReturnValue method in this example configures Siebel Open UI to use the same flow to run this code that it uses when it makes a synchronous call in a connected environment. For information about these methods, see [“callback Method” on page 419](#) and [“setReturnValue Method” on page 418](#).

- 4 If Siebel Open UI must run the call to any asynchronous method from a business service method more than one time, then use the eachAsyncOp method.

For example, assume you located the following code in [Step 2](#):

```
business_service.prototype.Submit = function () {
    var n;
    for(var i =0; i<n; i++){
        // Code that goes into preExecute
        bc.SetFieldValue(fieldName[i], fieldValue[i]);
        // code that goes into postExecute
    }
    retObj = bc.WriteRecord();
    if(!retObj.err){
        //Call code here that sets the return value to bRet
        return ({err: false, retVal: bRet});
    }
    else{
        return ({err: true});
    }
};
```

In this example, you replace the code that you located in [Step 2](#) with the following code:

```
PharmaCallSubmitsvc.prototype.Submit = function () {
    var preExecutecall= function(i){
        args[0]=fieldName[i]; //send the args while calling the async operation which
        is bc.SetFieldValue here
        args[1]=fieldData[i];
    }
    $.eachAsyncOp(preExecutecall, n, function(){
        //Code here that sets the return value to bRet
        $.setReturnValue({err: false, retVal: bRet});
    });
};
```



```

    return args; // arguments returned must always be in an array
}
var postExecuteCall = function(retObj){
    if(!retObj.err){
        //Do something using the return value obtained from the async call
    }
}
var configObj =
{execute: bc.SetFieldValue, preExecute: preExecuteCall, postExecute: postExecuteCall, iterations: n, executeScope: bc};
$.eachAsyncOp(this, configObj);
$.callBack(this, function(returnObj){
    if(!retObj.err){
        bc.WriteRecord();
        $.callBack(this, function(retObj){
            if(!retObj.err){
                //Call code here that sets the return value to bRet
                $.setReturnValue({err: false, retVal: bRet});
            }
        });
    }
});
};

```

In this example, Siebel Open UI uses the `eachAsyncOp` method to call the `SetFieldValue` method more than one time to set the value for more than one field. For information about these methods, see [“eachAsyncOp Method” on page 419](#) and [“SetFieldValue Method” on page 404](#), and [“WriteRecord Method” on page 407](#).

Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects

This topic describes how to use a Siebel business service or a JavaScript service to customize a predefined, Siebel CRM applet or business component.

Customizing Predefined Business Components

The example in this topic describes how to register and call a custom JavaScript method that customizes a predefined business component. You must configure Siebel Open UI to register a custom method before Siebel Open UI can call it.

To customize predefined business components

- 1 Use a JavaScript editor to create a new JavaScript file.

2 Specify the input properties that Siebel Open UI must send to the ServiceRegistry method.

The ServiceRegistry method uses input properties to register your custom method. For more information, see [“Properties You Must Include to Register Custom Business Services” on page 412](#).

You add the following code:

- a** Create the namespace for the JavaScript class. In this example, you create a namespace for the pharmacallsvc class:

```
if (typeof (Siebel App. pharmacallsvc) === "undefined") {  
    Siebel JS. Namespace(' Siebel App. pharmacallsvc' );
```

- b** Define the variables:

```
var oconsts = Siebel App. Offlineconstants;  
var inputObj = {};
```

- c** Specify the business component where Siebel Open UI applies your customization. In this example, you specify the Pharma Professional Call - Mobile business component:

```
inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Professional Call -  
Mobile";
```

- d** Specify the type of object that you are customizing. In this example, you are customizing a business component:

```
inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =  
oconsts.get("DOUI REG_OBJ_TYPEBUSCOMP");
```

- e** Specify the name of the predefined method that you are customizing. In this example, you are customizing the WriteRecord method:

```
inputObj [oconsts.get("DOUI REG_OBJ_MTHD")] = "WriteRecord";
```

- f** Specify the name of the JavaScript class where the method you are customizing resides. In this example, this method resides in the pharmacallsvc class:

```
inputObj [oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacallsvc";
```

- g** Specify the name of the custom service method that contains the customization of the WriteRecord method:

```
inputObj [oconsts.get("DOUI REG_SRVC_MTDH")] = "WriteRecord";
```

- h** Specify the type of customization:

```
inputObj [oconsts.get("DOUI REG_EXT_TYPE")] =  
oconsts.get("DOUI REG_EXT_TYPEPRE");
```

3 Register the custom JavaScript method that you specified in [Step 2](#). This code calls the ServiceRegistry method:

```
Siebel App. S_App. GetModel (). ServiceRegistry(inputObj);
```

4 Define the constructor:

```
Siebel App. pharmacallsvc = (function () {  
    function pharmacallsvc() {  
    }  
})
```

5 Extend the custom JavaScript class:

```
Siebel JS. Extend(pharmacallsvc, SiebelApp.ServiceModel);
```

6 Specify the custom WriteRecord method:

```
pharmacallsvc.prototype.WriteRecord = function (psInputArgs) { //get the inputs
var psOutArgs = SiebelApp.S_App.NewPropertySet();
return psOutArgs; //return the outputs
};
return pharmacallsvc;
} ();
}
```

The custom method must include your customization logic. This code gets the property set from the predefined WriteRecord method and uses it as input to your custom WriteRecord method. The custom WriteRecord method then returns an output property set to the predefined WriteRecord method.

The following code is the completed code for this topic:

```
if (typeof (SiebelApp.pharmacallsvc) === "undefined") {
Siebel JS.Namespace(' SiebelApp.pharmacallsvc' );
var oconsts = SiebelApp.Offlineconstants;
var inputObj = {};
inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Professional Call - Mobile";
inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =
oconsts.get("DOUI REG_OBJ_TYPEBUSCOMP");
inputObj [oconsts.get("DOUI REG_OBJ_MTHD")] = "WriteRecord";
inputObj [oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacallsvc";
inputObj [oconsts.get("DOUI REG_SRVC_MTDH")] = "WriteRecord";
inputObj [oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE");
SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
SiebelApp.pharmacallsvc = (function () {
function pharmacallsvc() {
}
Siebel JS. Extend(pharmacallsvc, SiebelApp.ServiceModel);
pharmacallsvc.prototype.WriteRecord = function (psInputArgs) { //get the inputs
var psOutArgs = SiebelApp.S_App.NewPropertySet();
return psOutArgs; //return the outputs
};
return pharmacallsvc;
} ();
}
```

Customizing Predefined Applets

The example in this topic registers a custom method that customizes a predefined applet. The work you do in this topic is very similar to the work you do in [“Customizing Predefined Business Components” on page 361](#). The only difference occurs when you specify the input object for the applet and the type of object.

To customize predefined applets

- Do [Step 1 on page 361](#) through [Step 6 on page 363](#) with the following differences:
 - For [Step c on page 362](#), specify the applet where Siebel Open UI applies your customization. In this example, you specify the Pharma Call Entry Mobile applet:

```
inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";
```

- For [Step d on page 362](#), specify the type of object that you are customizing. You specify an applet instead of a business component:

```
inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =  
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");
```

The following code is the completed code for this topic:

```
if (typeof (Siebel App. pharmacallsvc) === "undefined") {  
  Siebel JS. Namespace(' Siebel App. pharmacallsvc' );  
  var oconsts = Siebel App. Offlineconstants;  
  var inputObj = {};  
  inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";  
  inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =  
  oconsts.get("DOUI REG_OBJ_TYPEAPPLET");  
  inputObj [oconsts.get("DOUI REG_OBJ_MTHD")] = "InvokeMethod";  
  inputObj [oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacallsvc";  
  inputObj [oconsts.get("DOUI REG_SRVC_MTDH")] = "InvokeMethod";  
  inputObj [oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE");  
  Siebel App. S_App. GetModel (). ServiceRegistry(inputObj );  
  Siebel App. pharmacallsvc = (function () {  
    function pharmacallsvc() {  
    }  
    Siebel JS. Extend(pharmacallsvc, Siebel App. ServiceModel );  
    pharmacallsvc.prototype.InvokeMethod = function (psInputArgs) { //get the inputs  
      var psOutArgs = Siebel App. S_App. NewPropertySet();  
      return psOutArgs; //return the outputs  
    };  
    return pharmacallsvc;  
  } ());  
}
```

Using Custom JavaScript Methods

The example in this topic describes how to call a custom JavaScript method that does not customize a predefined method. Siebel Open UI does not require you to register a custom JavaScript method. Instead, you configure Siebel Open UI to do the following work:

- Override the InvokeMethod to call your custom method.
- Override the CanInvokeMethod method to enable or disable your custom method.

The `offline_predefined_js_call_example.js` file contains the code that this example describes. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

To use custom JavaScript methods

- 1 Use a JavaScript editor to create a new JavaScript file.
- 2 Register the InvokeMethod and CanInvokeMethod methods. You add the following code:

```
if (typeof (Siebel App. pharmacalI svc) === "undefined") {
    Siebel JS. Namespace(' Siebel App. pharmacalI svc');
    var inputObj = {};
    var oconsts = Siebel App. Offl ineconstants;
    inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";
    inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");
    inputObj [oconsts.get("DOUI REG_OBJ_MTHD")] = "CanI nvokeMethod";
    inputObj [oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacalI svc";
    inputObj [oconsts.get("DOUI REG_SRVC_MTDH")] = "CanI nvokeMethod";
    inputObj [oconsts.get("DOUI REG_EXT_TYPE")] =
oconsts.get("DOUI REG_EXT_TYPEPRE");
    Siebel App. S_App. GetModel (). Servi ceRegi stry(inputObj );
    inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";
    inputObj [oconsts.get("DOUI REG_OBJ_TYPE")] =
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");
    inputObj [oconsts.get("DOUI REG_OBJ_MTHD")] = "I nvokeMethod";
    inputObj [oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacalI svc";
    inputObj [oconsts.get("DOUI REG_SRVC_MTDH")] = "I nvokeMethod";
    inputObj [oconsts.get("DOUI REG_EXT_TYPE")] =
oconsts.get("DOUI REG_EXT_TYPEPRE");
    Siebel App. S_App. GetModel (). Servi ceRegi stry(inputObj );
    Siebel App. pharmacalI svc = (function () {
        function pharmacalI svc(pm) {
        }
        Siebel JS. Extend(pharmacalI svc, Siebel App. Servi ceModel ); //Extendi ng
        pharmacalI svc. prototype. InvokeMethod = function (psInputArgs) {
            var svcMthdName = "";
            var psOutArgs = Siebel App. S_App. NewPropertySet();
```

For more information about this code, see the description about the inputObj argument in [“ServiceRegistry Method” on page 411](#). Also see [“CanInvokeMethod Method” on page 388](#) and [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#).

- 3 Get the value of the MethodName argument from the psInputArgs method:

```
svcMthdName = psInputArgs.GetProperty("MethodName").toStri ng();
```

- 4 Call the Submit method:

```
if (svcMthdName === "Submi t") {
    thi s. Submi t();
    $. cal lback(thi s, functi on (retObj) {
```

For information, see [“callback Method” on page 419](#).

- 5 Do one of the following:

- If InvokeMethod handles the submit call that you define in [Step 4](#), then you use the following code to set the Invoked property to true:

```
    if (!retObj.err) {
        psOutArgs.SetProperty("Invoked", true);
        $.setReturnValue({err: "", retVal: psOutArgs});
    }
    else {
        psOutArgs.SetProperty("Invoked", true);
        $.setReturnValue({err: retObj.err, retVal: psOutArgs});
    }
    });
}
```

For information see [“setReturnValue Method” on page 418](#).

- If InvokeMethod does not handle the submit call that you define in [Step 4](#), then you must use the following code to configure Siebel Open UI to set the Invoked property to false. This code is required for any InvokeMethod method that you configure Siebel Open UI to override:

```
    else {
        psOutArgs.SetProperty("Invoked", false);
        $.setReturnValue({err: "", retVal: psOutArgs});
    }
    return(psOutArgs);
};
```

- If the current, overridden CanInvokeMethod method handles the submit call that you define in [Step 4](#), then you must set the Invoked property to true. Siebel Open UI includes the return value in the RetVal property for the method from CanInvokeMethod. You can set this method according to your requirements:

```
pharmacalIsvc.prototype.CanInvokeMethod = function (psInputArgs) {
    var psOutArgs = SiebelApp.S_App.NewPropertySet();
    var svcMthdName = "";
    svcMthdName = psInputArgs.GetProperty("MethodName").toString();
    if (svcMthdName === "Submit") {
        psOutArgs.SetProperty("Invoked", true);
        psOutArgs.SetProperty("RetVal", true);
        $.setReturnValue({err: "", retVal: psOutArgs});
    }
}
```

For more information about RetVal, see [“setReturnValue Method” on page 418](#).

- 6 If the current, overridden CanInvokeMethod method does not handle the submit call, then use the following code to set the Invoked property to false:

```
    else {
        psOutArgs.SetProperty("Invoked", false);
        psOutArgs.SetProperty("RetVal", false);
        $.setReturnValue({err: "", retVal: psOutArgs});
    }
    return(psOutArgs);
};
pharmacalIsvc.prototype.Submit = function (psInputArgs) {
    var psOutArgs = SiebelApp.S_App.NewPropertySet();
    return(psOutArgs);
}
```

```
};
return pharmacallsvc;
} ();
```

Using Custom Siebel Business Services

This topic describes how to call a Siebel business service that you customize. You must configure Siebel Open UI to register this business service before Siebel Open UI can call it.

To use custom Siebel business services

- 1 Use a JavaScript editor to create a new JavaScript file.
- 2 Register your custom business service. You add the following code:

```
var inputObj = {};
inputObj[oconsts.get("DOUI REG_OBJ_NAME")] = "business_service";
inputObj[oconsts.get("DOUI REG_SRVC_NAME")] = "class";
SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
```

where:

- *business_service* identifies the name of a custom business service.
- *class* identifies the JavaScript class that the custom business service references.

For example:

```
if (typeof (SiebelApp.PharmaCallValidatorsvc) === "undefined") {
    SiebelJS.Namespace('SiebelApp.PharmaCallValidatorsvc');

    var oconsts = SiebelApp.Offlineconstants;
    var inputObj = {};

    inputObj[oconsts.get("DOUI REG_OBJ_NAME")] = "LS Pharma Validation Service";
    inputObj[oconsts.get("DOUI REG_SRVC_NAME")] = "PharmaCallValidatorsvc";
    SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
    SiebelApp.PharmaCallValidatorsvc = (function () {

        function PharmaCallValidatorsvc() {
            SiebelApp.PharmaCallValidatorsvc.superclass.constructor.call(this);
        }
        SiebelJS.Extend(PharmaCallValidatorsvc, SiebelApp.ServiceModel);
    })();
}
```

For more information about the methods that this step uses, see the following topics:

- [“Properties You Must Include to Register Custom Business Services” on page 412](#)
- [“ServiceRegistry Method” on page 411](#)
- [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#)

- 3** Use `CanInvokeMethod` to determine if Siebel Open UI can call your custom business service method.

For example, the following code determines if Siebel Open UI can call the `CallValidate` business service method:

```
PharmaCallValidateSvc.prototype.CanInvokeMethod = function (svcMthdName) {
  if (svcMthdName === "CallValidate") {
    $.setReturnValue({err: "", retVal: true});
    return;
  }
  else {
    return
    SiebelApp.PharmaCallValidateSvc.superclass.CanInvokeMethod.call(this,
    svcMthdName);
  }
};
```

For more information about the methods that this step uses, see [“CanInvokeMethod Method” on page 388](#).

- 4** Use `InvokeMethod` to call your custom business service method.

For example, the following code calls the `CallValidate` business service method:

```
PharmaCallValidateSvc.prototype.InvokeMethod = function (svcMthdName,
psiNpargs) {
  var psOutArgs = SiebelApp.S_App.NewPropertySet();
  if (!svcMthdName) {
    $.setReturnValue({err: "", retVal: true});
    return;
  }
  if (svcMthdName === "CallValidate") {
    this.CallValidate(psiNpargs);
    $.callback(this, function(retObj){
      psOutArgs = retObj.retVal;
      this.CleanUp();
      $.setReturnValue({err: false, retVal: psOutArgs});
      return;
    });
  }
  else {
    return
    SiebelApp.PharmaCallValidateSvc.superclass.InvokeMethod.call(this,
    svcMthdName, psiNpargs);
  }

  PharmaCallValidateSvc.prototype.CallValidate = function (psiNpropset) {
    var psOutArgs = SiebelApp.S_App.NewPropertySet();
    //Some Logic
    $.setReturnValue({err: false, retVal: psOutArgs});
  };
};
```



```
        return PharmaCallValidatorsvc;  
    } ();  
}
```

For more information about the methods that this step uses, see the following topics:

- [“Invoke Method for Business Services” on page 410](#)
- [“InvokeMethod Method for Applets” on page 389](#)
- [“setReturnValue Method” on page 418](#)
- [“callback Method” on page 419](#)

Configuring Data Filters

It is recommended that you configure filters to reduce the amount of business component data that Siebel Open UI must download to do offline operations. Siebel Open UI comes predefined with a number of data filters. You can modify these filters. For more information about how to modify them, see the chapter about working with data filters in *Siebel Disconnected Mobile Applications Guide*.

Configuring Objects That Siebel Open UI Does Not Display in Clients

The Handheld Business Service only downloads fields, business component data, and business object data that Siebel Open UI displays in the client. You must configure Siebel Open UI to download these objects that it does not display in the client. To do this, you use the Settings tab of the Mobile Application view in the Administration - Siebel Remote screen in the administrative client. For more information, see the topic about configuring application settings in *Siebel Disconnected Mobile Applications Guide*.

Configuring Error Messages for Disconnected Clients

This topic describes how to configure Siebel Open UI to use the SetErrMsg method in your custom code to return and display a custom error message in a disconnected client.

To configure error messages for disconnected clients

- 1 Use an editor to open the file that calls a custom applet, business component, or business service.

This is the same file that you create in [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#).

- 2 Locate the code that might return an error message.

For example, assume your deployment includes the following code, and that this code calls a method that might return an error message:

```
BusComp.prototype.Caller = function ()  
    this.Called();  
    $.callback(this, function(retObj){
```

In this example, the Called method might return an error message. It calls the Caller method. These methods might reside in different locations in a production environment.

- 3** Add the following code to the code that you located in [Step 2](#):

```
    //Check for any errors  
    if(retObj.err){  
        $.setReturnValue(retObj);  
    }  
    else{  
        //Positive case  
        $.setReturnValue({err: false, retVal: false});  
    }  
});  
return;  
}
```

This code determines whether or not the Called method returns an error message. If it:

- Returns an error message, then this code calls the setReturnValue method.
- Does not return an error message, then the following code sets the err return value to null:

```
$.setReturnValue({err: false, retVal: false});
```

For information see [“setReturnValue Method” on page 418](#).

- 4** Add the following code to the code that you located in [Step 3](#):

```
BusComp.prototype.Called = function (){  
    var errParamArray = [];  
    errParamArray.push(value1, valueN);  
    SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("messageKey", errParamArray);  
    $.setReturnValue({err: "AppropriateErrorCode", retVal: false});  
}
```

where:

- *value1* is a property that Siebel Open UI sends to the SetErrorMsg method. You can configure Siebel Open UI to send up to eight properties.
- *messageKey* is a key that Siebel Open UI maps to the message string that it displays.

For more information, see [“SetErrorMsg Method” on page 420](#).

In this example, the following code calls the SetErrorMsg method:

```
SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("AppropriateErrorCode",  
errParamArray);
```

The following code makes sure Siebel Open UI returns an err value. The err value contains the error code:

```
$.setReturnValue({err: "AppropriateErrorCode", retVal: false});  
return;
```

The following code is the completed code that this example uses:

```
BusComp.prototype.Caller = function ()
{
    this.Called();
    $.callback(this, function(retObj){
        //Check for any errors
        if(retObj.err){
            $.setReturnValue(retObj);
        }
        else{
            //Positive case
            $.setReturnValue({err: false, retVal: false});
        }
    });
    return;
}
BusComp.prototype.Called = function (){
    var errParamArray = [];
    errParamArray.push(fieldName);
    SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("ErrorCode", errParamArray);
    $.setReturnValue({err: "AppropriateErrorCode", retVal: false});
    return;
}
```

where:

- *ErrorCode* identifies a messageKey. Siebel Open UI gets the message text for the message key from the swemessages_*language_code*.js file that resides in a local folder. For example, swemessages_enu.js.
- *fieldName* identifies the name of a business component field. This field contains the values that Siebel Open UI displays in the error message. For example, the predefined BCErrNoSuchField message key includes the following message text in the swemessages_enu.js file:

"Field '%1' not found in BusComp."

SetErrorMsg replaces %1 with the value that Siebel Open UI passes in the errParamArray. For example:

```
errParamArray.push("Name");
SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("BCErrNoSuchField", errParamArray)
```

In this example, Siebel Open UI replaces "%1" with the value Name:

"Field 'Name' not found in BusComp."

Customizing Siebel Pharma for Siebel Mobile Disconnected Clients

This topic includes an example of customizing Siebel Pharma in Siebel Open UI for display in a Siebel Mobile disconnected client. For more information about the functionality that these customizations modify, see the chapter that describes how to use the Siebel Mobile Disconnected Application for Siebel Pharma in *Siebel Disconnected Mobile Applications Guide*.

This topic customizes Siebel Pharma to submit a Pharma Call record depending on whether or not Siebel Open UI already submitted this call. It makes sure Siebel Open UI does not overwrite a call that it already submitted to the Siebel Server. To submit a call in Siebel Pharma, the user must do the following work:

- Enter all information for the call.
- Add at least one sample for the call.
- Get the required signature for the samples that the call includes.
- Set the status for the call to Planned or Signed.
- Tap Submit.

Siebel Pharma locks a call after it submits this call, and then the user can no longer edit or update the call. You can modify some of this behavior. For more information about the work you do in this topic, see [“Process of Customizing Siebel Open UI for Siebel Mobile Disconnected” on page 355](#). For more information about the methods that this example uses, see [“Methods You Can Use to Customize Siebel Mobile Disconnected” on page 387](#).

To customize Siebel Pharma for Siebel Mobile Disconnected clients

- 1 Create a new JavaScript file.

You can use any file name that is meaningful to your deployment. For example, you can use a short name that indicates what the business service accomplishes. It is recommended that the file name end with `svc.js` or `service.js`. For example, `callsvc.js`. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support. For more information about the folders you can use to store your customizations, see [“Organizing Files That You Customize” on page 122](#).

- 2 Register an asynchronous business component method. You add the following code to the file you created in [Step 1](#):

```
var inputArgs = {};  
var oconsts = SiebelApp.OfferConstants;  
var childBCArrObjs = [];  
inputArgs[oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";  
inputArgs[oconsts.get("DOUI REG_OBJ_TYPE")] =  
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");  
inputArgs[oconsts.get("DOUI REG_OBJ_MTHD")] = "CanInvokeMethod";  
inputArgs[oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacallsvc";  
inputArgs[oconsts.get("DOUI REG_SRVC_MTDH")] = "CanInvokeMethod";  
inputArgs[oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE");  
SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);  
inputArgs[oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";
```

```
inputArgs[oconsts.get("DOUI REG_OBJ_TYPE")] =
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");
inputArgs[oconsts.get("DOUI REG_OBJ_MTHD")] = "InvokeMethod";
inputArgs[oconsts.get("DOUI REG_SRVC_NAME")] = "pharmacallsvc";
inputArgs[oconsts.get("DOUI REG_SRVC_MTDH")] = "InvokeMethod";
inputArgs[oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE");
SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);
```

This code registers the `CanInvokeMethod` method of the `pharmacallsvc` business service with the `CanInvokeMethod` of the `Pharma Call Entry Mobile` business component. It configures Siebel Open UI to call `CanInvokeMethod` of the `pharmacallsvc` business service every time it calls `CanInvokeMethod` on the business component. For more information, see [“ServiceRegistry Method” on page 411](#) and [“CanInvokeMethod Method” on page 388](#).

- 3 Add the following code immediately after the code you added in [Step 2](#):

```
SiebelApp.pharmacallsvc = (function () {
    function pharmacallsvc(pm) {
    }
    SiebelJS.Extend(pharmacallsvc, SiebelApp.ServiceModel);
})();
```

This code adds the `pharmacallsvc` method to the `pharmacallsvc` business service.

- 4 Specify the logic for your asynchronous method. You add the following code immediately after the code you added in [Step 3](#):

```
pharmacallsvc.prototype.CanInvokeMethod = function (psInputArgs) {
    var psOutArgs = SiebelApp.S_App.NewPropertySet();
    var svcMthdName = "";
    var pBusComp = this.GetContext().BusComp();
    svcMthdName = psInputArgs.GetProperty("MethodName").toString();

    if (svcMthdName === "Submit") {
        pBusComp.GetFieldValue("Call Status");
        $.callback(this, function (retObj) {
            var callStatus = retObj.retVal;
            if (callStatus !== "Submitted") {
                psOutArgs.SetProperty("Invoked", true);
                psOutArgs.SetProperty("RetVal", true);
                $.setReturnValue({err: false, retVal: psOutArgs});
            }
        });
    } else {
        psOutArgs.SetProperty("Invoked", true);
        psOutArgs.SetProperty("RetVal", false);
        $.setReturnValue({err: false, retVal: psOutArgs});
    }
};
```

This code defines `CanInvokeMethod`. This method determines whether or not Siebel Open UI can call a method in the current context of the business component. In this example, if the value for the active call record is:

- **Not submitted.** CanInvokeMethod returns a value of true, and this code sets the properties to call the CanInvokeMethod of the business service.
- **Submitted.** CanInvokeMethod returns a value of false, and this code sets the properties to not call the CanInvokeMethod of the business service.

This code uses the svcMthdName variable to send a value that indicates whether or not Siebel Open UI submitted the record. It sends this value to the code that you define in [Step 5](#).

For information about the methods that this step uses, see [“setReturnValue Method” on page 418](#), [“GetFieldValue Method” on page 397](#), and [“callback Method” on page 419](#).

- 5 Add the following code immediately after the code you added in [Step 4](#):

```
pharmacal | svc.prototype.InvokeMethod = function (psInputArgs) {  
    var svcMthdName = "";  
    var psOutArgs = SiebelApp.S_App.NewPropertySet();  
    svcMthdName = psInputArgs.GetProperty("MethodName").toString();  
    if (svcMthdName === "Submit") {  
        this.Submit();  
        $.callback(this, function (retObj) {  
            psOutArgs.SetProperty("Invoked", true);  
            $.setReturnValue({err: false, retVal: psOutArgs});  
        });  
    }  
};
```

This code configures Siebel Open UI to run InvokeMethod on the business service if the svcMthdName variable that you defined in [Step 4](#) contains a value of Submit.

- 6 Define the method that includes your customization logic. You add the following code immediately after the code you added in [Step 5](#):

```
pharmacal | svc.prototype.Submit = function () {  
    var model = SiebelApp.S_App.GetModel();  
    var pBusObj = model.GetBusObject("boName");  
    var pBusComp = pBusObj.GetBusComp("bcName");  
    var now = new Date();  
    var strStatusField = pBusComp.GetUserProperty("Status Field");  
    var pickName =  
        SiebelApp.S_App.GetActiveView().GetActiveApplet().GetControl("Status").  
        GetPickApplet();  
    pBusComp.SetFieldValue(strStatusField, "submit", true);  
    $.callback(this, function (retObj) {  
        pBusComp.WriteRecord();  
        $.callback(this, function (retObj) {  
        });  
    });  
};
```

This code defines the Submit method. It sets the value for the Status field to Submitted. It uses the following methods:

- BusComp. For information, see [“BusComp Method for Applets” on page 388](#).
- SetFieldValue. For information, see [“SetFieldValue Method” on page 404](#).

- WriteRecord. For more information, see [“WriteRecord Method” on page 407](#).

For more information about how this code uses the callback method, see [“Coding Callback Methods” on page 375](#).

7 Test your modifications:

- a** Tap Calls on the application banner to display the Calls list.
- b** Tap a call in the list that you know you have not submitted, and then tap Submit to submit the call.
- c** Verify that Siebel Open UI does the following:
 - Modifies the call status to Submitted.
 - Locks the call
 - Decreases the sample inventory for the sales representative according to the samples and promotional items that the call dropped off
 - Closes the call.
 - Allows you to review, but not edit the call details.
- d** Tap a call in the list that you know you have already submitted, and then tap Submit to submit the call.

Make sure Siebel Open UI does not overwrite this call. Make sure it displays a dialog box that describes that you have already submitted this call.

Coding Callback Methods

The code in [Step 6 on page 374](#) is an example of how to code a callback method in an asynchronous environment. For example, it uses the SetFieldValue and the WriteRecord methods, which are asynchronous methods, rather than the SetCommitPending method, which is a synchronous method. A *callback method* is a type of JavaScript method that Siebel Open UI sends to method A as a property, and then calls this callback method from method A, where A is any JavaScript method.

If you configure Siebel Open UI to call an asynchronous method, then it is recommended that the next line of code that occurs after this call include a callback method. For example, the following code from [Step 4 on page 373](#) uses a callback method to handle the asynchronous GetFieldValue method:

```
pBusComp.GetFieldValue("Call Status");
$.callback(this, function (retObj) {
    var callStatus = retObj.retVal;
    if(callStatus !== "Submitted"){
```

For more information, see [“GetFieldValue Method” on page 397](#) and [“callback Method” on page 419](#).

Customizing Siebel Service for Siebel Mobile Disconnected Clients

This topic includes some examples that describe how to customize Siebel Service in Siebel Open UI for a Siebel Mobile disconnected client. It includes the following information:

- [“Allowing Users to Commit Part Tracker Records” on page 376](#)
- [“Allowing Users to Return Parts” on page 378](#)
- [“Allowing Users to Set the Activity Status” on page 384](#)

For more information about:

- Work you do in this topic, see [“Process of Customizing Siebel Open UI for Siebel Mobile Disconnected” on page 355](#)
- Methods that these examples use, see [“Methods You Can Use to Customize Siebel Mobile Disconnected” on page 387](#)
- Functionality that these customizations modify, see the chapter that describes how to use the Siebel Mobile Disconnected Application for Siebel Service in *Siebel Disconnected Mobile Applications Guide*

Allowing Users to Commit Part Tracker Records

The example in this topic describes how to enable the Commit button so that users can commit a Part Tracker record. To set the Commit Flag for a Part Tracker record, the user navigates to the Activities - Part Tracker view, chooses a Part Tracker record, and then clicks Commit. If the part is:

- Not already committed, then Siebel Open UI commits the part.
- Already committed, then Siebel Open UI displays a message that the part is already committed.

To allow users to commit Part Tracker records

- 1 In Windows Explorer, navigate to the following folder:

```
CLIENT_HOME\appweb\PUBLIC\language_code\release_number\scripts\siebel\offline
```

- 2 Copy the servicecommitpartconsumed.js file to the following folder:

```
CLIENT_HOME\SWEApp\PUBLIC\language_code\files\custom\
```

For more information, see [“Organizing Files That You Customize” on page 122](#).

- 3 Use a JavaScript editor to open the file you created in [Step 2](#).
- 4 Locate the following code that resides near the beginning of the file:

```
if (typeof (Siebel App. commitpartconsumed) === "undefined") {  
  Siebel JS.Namespace(' Siebel App. commitpartconsumed');
```

- 5 Add the following code immediately after the code that you located in [Step 4](#):

```
var inputArgs = {};  
var oconsts = Siebel App. Offlineconstants;  
inputArgs[oconsts.get("DOUI REG_OBJ_NAME")] = "SHCE Service FS Activity Part  
Movements List Applet - Mobile";  
inputArgs[oconsts.get("DOUI REG_OBJ_TYPE")] =  
oconsts.get("DOUI REG_OBJ_TYPEAPPLET");  
inputArgs[oconsts.get("DOUI REG_OBJ_MTHD")] = "CommitPartMvmtClient";
```



```
inputArgs[oconsts.get("DOUI REG_SRVC_NAME")] = "commitpartconsumed";
inputArgs[oconsts.get("DOUI REG_SRVC_MTDH")] = "CommitPartMvmtClient";
inputArgs[oconsts.get("DOUI REG_EXT_TYPE")] = null;
SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);
```

This code registers the service. For more information, see [“ServiceRegistry Method” on page 411](#).

- 6 Add the following CanInvokeMethod method immediately after the code that you added in [Step 5](#):

```
commitpartconsumed.prototype.CanInvokeMethod = function (svcMthdName) {
  if (svcMthdName === "CommitPartMvmtClient") {
    return true;
  }
  else
    return SiebelApp.commitpartconsumed.superclass.CanInvokeMethod.call(
      this, svcMthdName);
};
```

This code determines whether or not Siebel Open UI can call a method in the current context of the business component.

- 7 Add the following InvokeMethod method immediately after the code that you added in [Step 6](#):

```
commitpartconsumed.prototype.InvokeMethod = function (svcMthdName, psi nparams) {
  var psOutArgs = SiebelApp.S_App.NewPropertySet();
  if (!svcMthdName) {
    return (false);
  }
  if (svcMthdName === "CommitPartMvmtClient") {
    psOutArgs = this.CommitPartMvmtClient();
  }
  else {
    return SiebelApp.commitpartconsumed.superclass.InvokeMethod.call(
      this, svcMthdName, psi nparams);
  }
  return (psOutArgs);
};
```

This code calls the CommitPartMvmtClient service method if the user clicks the Commit button.

- 8 Add the following code immediately after the code that you added in [Step 7](#):

```
commitpartconsumed.prototype.CommitPartMvmtClient = function () {
  SiebelJS.Log('Invoked CommitPartMvmtClient Method. ');
  var pServiceNVBC;
  var cszCommitFlag;
  var pModel;
  pModel = SiebelApp.S_App.Model;
  var pServiceNVBO = pModel.GetBusObject("boName");
  pServiceNVBC = pServiceNVBO.GetBusComp("bcName");
  cszCommitFlag = pServiceNVBC.GetFieldValue("Commit Txn Flag");
  if (cszCommitFlag === 'Y'){
    SiebelJS.Log('Consumed Part Is Already In Committed State');
  }
  else
  {
```

```
// pService.nvBC.ActivateField("Commit Txn Flag");
//pService.nvBC.UpdateRecord();
pService.nvBC.SetFieldValue("Commit Txn Flag", "Y", true);
pService.nvBC.WriteRecord();
}
};
```

This code determines whether or not the record is already committed. The `DoInvoke` method calls the `CommitPartMvmtClient` method, and then the `CommitPartMvmtClient` method examines the value of the `Commit Txn Flag` field. If this value is:

- **Y.** Siebel Open UI has already committed the record and displays a Consumed Part Is Already In Committed State message.
- **N.** Siebel Open UI has not committed the record and writes the record to the local database.

For more information about the methods that this code uses, see [“GetFieldValue Method” on page 397](#), [“SetFieldValue Method” on page 404](#), and [“WriteRecord Method” on page 407](#).

Allowing Users to Return Parts

The example in this topic describes how to enable the RMA button so that a user can return a part. To return a part, the user creates a part tracker record, and then clicks the RMA button to create a Return Material Authorization (RMA) record. The work you do to allow a user to return a part is similar to the work you do to allow a user to commit a Part Tracker record. For example, registering the service, calling the `CanInvoke` method, `DoInvoke` method, and so on.

You add the code that specifies how to do the RMA return in step [Step 4 on page 378](#) through [Step 10 on page 383](#). The `rma_return.js` file contains this code. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

To allow users to return parts

- 1 In Windows Explorer, navigate to the following folder:

```
CLIENT_HOME\appweb\PUBLIC\language_code\release_number\scripts\siebel\offline
```

- 2 Use a JavaScript editor to open the `servicecmtparts.js` file.
- 3 Add the following code to the `InvokeMethod` method:

```
var model = SiebelApp.S_App.GetModel();
var pBusObj = model.GetBusObject("boName");
var pBusComp = pBusObj.GetBusComp("bcName");
```

This code gets the active business component for the applet that displays the RMA button.

- 4 Add the following code. This code declares the objects:

```
if (typeof (SiebelApp.committpartconsumed) === "undefined") {
    SiebelJS.Namespace('SiebelApp.committpartconsumed');

    var inputArgs = {};
    var oconsts = SiebelApp.Offlineconstants;
```

```

    inputArgs[oconsts.get("DOUI REG_OBJ_NAME")]="SHCE Service FS Activity Part
Movements List Applet - Mobile";
    inputArgs[oconsts.get("DOUI REG_OBJ_TYPE")] = oconsts.get("DOUI REG_OBJ_TYPEAPPLE
T");
    inputArgs[oconsts.get("DOUI REG_OBJ_MTHD")]="CanI nvokeMethod";
    inputArgs[oconsts.get("DOUI REG_SRVC_NAME")]="commi tpartconsumed";
    inputArgs[oconsts.get("DOUI REG_SRVC_MTDH")]="CanI nvokeMethod";
    inputArgs[oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE")
;
    Siebel App. S_App. GetModel (). Servi ceRegi stry(i nputArgs);

    inputArgs={};

    inputArgs[oconsts.get("DOUI REG_OBJ_NAME")]="SHCE Service FS Activity Part
Movements List Applet - Mobile";
    inputArgs[oconsts.get("DOUI REG_OBJ_TYPE")] = oconsts.get("DOUI REG_OBJ_TYPEAPPLE
T");
    inputArgs[oconsts.get("DOUI REG_OBJ_MTHD")]="I nvokeMethod";
    inputArgs[oconsts.get("DOUI REG_SRVC_NAME")]="commi tpartconsumed";
    inputArgs[oconsts.get("DOUI REG_SRVC_MTDH")]="I nvokeMethod";
    inputArgs[oconsts.get("DOUI REG_EXT_TYPE")] = oconsts.get("DOUI REG_EXT_TYPEPRE")
;

    Siebel App. S_App. GetModel (). Servi ceRegi stry(i nputArgs);

    inputArgs={};

```

For information about the methods that this code uses, see the following topics:

- [“CanInvokeMethod Method” on page 388](#)
- [“ServiceRegistry Method” on page 411](#)
- [“InvokeMethod Method for Applets” on page 389](#)

5 Add the following code. This code calls the CanInvokeMethod method:

```

Siebel App. commi tpartconsumed = (function () {
    functi on commi tpartconsumed(pm) {
    }
    var commi tObj = new commi tpartconsumed();
    commi tpartconsumed.prototype.CanI nvokeMethod = functi on (psI nputArgs) {
        var psOutArgs = Siebel App. S_App. NewPropertySet();
        var svcMthdName = "";
        svcMthdName = psI nputArgs. GetProperty("Method Name"). toStri ng();
        i f (svcMthdName === "Commi tPartMvmtCl ient") {
            psOutArgs. SetProperty("I nvokeMethod", true);
            psOutArgs. SetProperty("RetVal ", true);
            $. setReturnVal ue({err: fal se, retVal : psOutArgs});
        }

        el se i f (svcMthdName === "OrderPartsRMA") {
            psOutArgs. SetProperty("I nvokeMethod", true);
            psOutArgs. SetProperty("RetVal ", true);
            $. setReturnVal ue({err: fal se, retVal : psOutArgs});
        }
    }
}

```

```
    }  
    else{  
        psOutArgs.SetProperty("Invoked", false);  
        psOutArgs.SetProperty("RetVal", false);  
        $.setReturnValue({err: false, retVal: psOutArgs});  
    }  
};
```

- 6 Add the following code. This code calls the InvokeMethod method:

```
commi tpartconsumed.prototype.InvokeMethod = function (psInputArgs) {  
    var svcMthdName = "";  
    var psOutArgs = Siebel App.S_App.NewPropertySet();  
    svcMthdName = psInputArgs.GetProperty("MethodName").toString();  
    if (svcMthdName === "CommitPartMvmtClient") {  
        this.CommitPartMvmtClient();  
        $.callBack(this, function(retObj){  
            psOutArgs.SetProperty("Invoked", true);  
            $.setReturnValue({err: false, retVal: psOutArgs});  
        });  
    }  
    else{  
        psOutArgs.SetProperty("Invoked", false);  
        $.setReturnValue({err: false, retVal: psOutArgs});  
    }  
    if (svcMthdName === "OrderPartsRMA") {  
        this.OrderPartsRMA();  
        $.callBack(this, function(retObj){  
            psOutArgs.SetProperty("Invoked", true);  
            $.setReturnValue({err: false, retVal: psOutArgs});  
        });  
    }  
    else{  
        psOutArgs.SetProperty("Invoked", false);  
        $.setReturnValue({err: false, retVal: psOutArgs});  
    }  
};
```

For information about the methods that this code uses, see [“setReturnValue Method” on page 418](#).

- 7 Add the code that gets values for the following fields:

- Product Id
- Product Name
- Used Quantity
- Id
- Status
- Asset Number
- Part Number

You add the following code:

```

commi tpartconsumed.prototype.createMAOrder = function (orderType) {
    var sOrderId;
    var cszOrderId;
    var sAssetNum;
    var sPartNum;
    var sStatus;
    var sProductId;
    var sProductName;
    var sQuantity;
    var sActi vi tyPartMvmtID;
    var pModel ;
    var pFSActi vi tyPartsMovementBC;
    var pActi onBC;
    var sSR_Id;
    var pServi ceRequestBC;
    var pOrderEntry_OrdersBC;
    var pOrderEntry_Li nel temBC;
    var errParamArray = [];
    pModel = Si ebel App. S_App. Model ;
    var pBusObj = pModel .GetBusObj ect("boName")
    pFSActi vi tyPartsMovementBC=pBusObj .GetBusComp("bcName");
    $.cal l back(thi s, functi on(retObj){
        sOrderId=retObj .retVal ;
        i f (uti l s. I sEmpty(sOrderId)){
            pFSActi vi tyPartsMovementBC. GetFi el dVal ue("");
            var oPsDR_Header: PropertySet = Si ebel App. S_App. NewPropertySet();
            // Cannot use the same property set i n GetMul ti pleFi el dVal ues, must use a
di fferent
            // one for the values. The process wi ll not error, but Si ebel Open UI wi ll
not place
            // the values i n the property set.
            var l PS_val ues: PropertySet = Si ebel App. S_App. NewPropertySet();
            oPsDR_Header. SetProperty("Product I d", "");
            oPsDR_Header. SetProperty("Used Quanti ty", "");
            oPsDR_Header. SetProperty("I d", "");
            oPsDR_Header. SetProperty("Asset Number", "");
            oPsDR_Header. SetProperty("Part Number", "");
            $.cal l back(thi s, functi on(retObj){
                sPartNum=retObj .retVal ;
                pActi onBC =
Si ebel App. S_App. GetActi veVi ew(). GetActi veAppl et(). BusComp(). ParentBuscomp();
                pActi onBC. GetFi el dVal ue("Acti vi ty SR I d");
                $.cal l back(thi s, functi on(retObj){
                    sSR_Id = retObj .retVal ;
                    i f(sSR_Id==""){
                        //Acti vi ty has no associated SR. . . Hence the operati on wi ll be aborted
                        Si ebel App. S_App. Offl i neErrorObj ect. SetErrorMsg("I DS_ERR_FS_MI SSI NG_
SR", errParamArray);
                        $.setReturnVal ue({err: "I DS_ERR_FS_MI SSI NG_SR", retVal: ""});
                        return;
                    }
                });
            });
        }
    });
}

```

```
    }
  });
}
```

For information about the methods that this code uses, see [“GetFieldValue Method” on page 397](#) and [“callback Method” on page 419](#).

8 Add the code that gets the parent business component and the following business components:

- Service Request
- Order Entry - Orders
- Order Entry - Line Items

This code also determines whether or not a service request is not associated with the activity. If not, then it aborts the operation. You add the following code:

```
else{
  pModel = Siebel App. S_App. Model ;
  pServiceRequestBC = pModel . BusObj ("Service Request"). BusComp("Service
Request");
  pOrderEntry_OrdersBC = Siebel App. S_App. Model . GetBusObj ("Service
Request"). BusComp("Order Entry - Orders");
  pOrderEntry_LineItemBC = pModel . BusObj ("Service Request"). BusComp("Order Entry
- Line Items");
  //CREATE ORDER Header.
  pOrderEntry_OrdersBC. ExecuteQuery();
  $. call back(this, function(retObj){
```

9 Add the code that creates the Order Header record and sets the field values. For example, for the Order Type field. You add the following code:

```
pOrderEntry_OrdersBC. NewRecord(true);
$. call back(this, function(retObj){
  sLocal eVal = Siebel App. S_App. Model . GetLovNameVal (orderType,
"FS_ORDER_TYPE");
  pOrderEntry_OrdersBC. SetFi el dVal ue("Order Type", sLocal eVal , true);
  $. call back(this, function(retObj){
    pOrderEntry_OrdersBC. Wri teRecord();
    $. call back(this, function(retObj){
      pOrderEntry_OrdersBC. GetFi el dVal ue("Id");
      $. call back(this, function(retObj){
        sOrderI temI d=retObj . retVal ;
        pOrderEntry_OrdersBC. GetFi el dVal ue("Id");
        $. call back(this, function(retObj){
          m_sOrderHeaderI d=retObj . retVal ;
          pOrderEntry_Li nel temBC. ExecuteQuery();
          $. call back(this, function(retObj){
```

For information about the methods that this code uses, see [“SetFieldValue Method” on page 404](#), [“WriteRecord Method” on page 407](#), [“NewRecord Method” on page 474](#).

- 10** Add the code that creates the order line item record, commits this record, and sets the value for the Order Item Id field in the active business component. This value is the row Id of the order header record that Siebel Open UI creates. This code sets the field value for each of the following fields:

- Product
- Quantity Requested
- Asset #
- Part #
- Product Status Code
- Order Header Id

You add the following code:

```
pOrderEntry_LineItemBC.NewRecord(true);
$.callBack(this, function(retObj){
    pOrderEntry_LineItemBC.SetFieldValue("Product Id", sProductId, true);
    $.callBack(this, function(retObj){
        pOrderEntry_LineItemBC.SetFieldValue("Product", sProductName, true);
        $.callBack(this, function(retObj){
            pOrderEntry_LineItemBC.SetFieldValue("Quantity Requested", sQuantity, true);
            $.callBack(this, function(retObj){
                if(!utils.isEmpty(sAssetNum)){
                    pOrderEntry_LineItemBC.SetFieldValue("Asset Number", sAssetNum, true);
                    $.callBack(this, function(retObj){
                        });
                }
                if(!utils.isEmpty(sPartNum)){
                    pOrderEntry_LineItemBC.SetFieldValue("Part Number", sPartNum, true);
                    $.callBack(this, function(retObj){
                        });
                }
                if(!utils.isEmpty(sStatus)){
                    pOrderEntry_LineItemBC.SetFieldValue("Product Status Code", sStatus, true);
                    $.callBack(this, function(retObj){
                        });
                }
                pOrderEntry_LineItemBC.GetFieldValue("Id");
                $.callBack(this, function(retObj){
                    sOrderItemID=retObj.retVal;
                    pOrderEntry_LineItemBC.SetFieldValue("Order Header Id", m_sOrderHeaderId, true);
                    $.callBack(this, function(retObj){
                        pOrderEntry_LineItemBC.WriteRecord();
                        $.callBack(this, function(retObj){
                            pFSActivityPartsMovementBC.SetFieldValue("Order Item Id", sOrderItemID, true);
```

```
$.callBack(this,function(retObj){
    pFSActivityPartsMovementBC.WriteRecord();
    $.callBack(this,function(retObj){
        });
    });
});
});
});
});
});
```

- 11** Save, and then close the servicemtparts.js file.
- 12** Test your modifications:
 - a** Log in to the disconnected client.
 - b** Click the Activities tab.
 - c** Create an activity, and then click Part Tracker.
 - d** Create a part tracker record.
 - e** Click the RMA button to create a Return Material Authorization (RMA) record.
 - f** Make sure Siebel Open UI creates the RMA record and displays the correct values in the fields of this record, such as the Product Id, Product Name, Used Quantity, Quantity Requested, Asset #, and so on.

Allowing Users to Set the Activity Status

The example in this topic describes how to enable the activity status so that the user can update this status during the service call life cycle. For example, a field service representative can examine an Activity that is set to Dispatched, set this status to Acknowledged to acknowledge that this representative examined the activity, set the status to EnRoute, travel to the customer site, set it to Arrive, set it to In Progress while working on the service call, and then set it to Finish after finishing the service call. Siebel Open UI includes the following status values:

- Dispatched
- Acknowledged
- Declined
- En Route
- Arrive
- In Progress
- Hold
- Resume

■ Finish

Siebel Open UI enables and disables the status depending on the current value of the status. For example, if the representative sets the status to Acknowledged, then Siebel Open UI allows the user to choose the EnRoute status and disables all other values.

The work you do to allow a user to set the status is similar to the work you do to allow a user to commit a Part Tracker record. For example, registering the service, and so on. For more information, see [“Allowing Users to Commit Part Tracker Records” on page 376](#).

To allow users to set the activity status

- 1 In Windows Explorer, navigate to the following folder:

`CLIENT_HOME\eappweb\PUBLIC\language_code\release_number\scripts\siebel\offline`

- 2 Use a JavaScript editor to open the serviceactstat.js file.

- 3 Locate the following code:

```
serviceactstat.prototype.InvokeSetActStatus=function(psl npArgs, svcMthdName){
    var psOutArgs=Siebel App. S_App. NewPropertySet();
    if(!psl npArgs){
        return (false);
    }
    if(psl npArgs.propArray.MethodName=="AcceptStatus")
        {psOutArgs=this.SetActi vi tyStatus("Acknowledged");
    }
    else if(psl npArgs.propArray.MethodName=="Start" || psl npArgs.propArray.
        MethodName=="Arri vedStatus"){psOutArgs=this.SetActi vi tyStatus("In
        Progress", "Arri vedStatus");
    }
    else if(psl npArgs.propArray.MethodName=="Decl i neStatus"){
        psOutArgs=this.SetActi vi tyStatus("Decl i ned");
    }
    else if(psl npArgs.propArray.MethodName=="EnrouteStatus"){
        psOutArgs=this.SetActi vi tyStatus("In Progress");
    }
    else if(psl npArgs.propArray.MethodName=="SuspendStatus"){
        psOutArgs=this.SetActi vi tyStatus("On Hold");
    }
    else if(psl npArgs.propArray.MethodName=="ResumeStatus"){
        psOutArgs=this.SetActi vi tyStatus("In Progress");
    }
    else if(psl npArgs.propArray.MethodName=="End" || psl npArgs.propArray.
        MethodName=="Fi ni shedStatus"){
        psOutArgs = this.SetActi vi tyStatus("Done", "Fi ni shedStatus");
    }
}
```

- 4 Add the following code immediately after the code you located in [Step 3](#):

```
serviceactstat.prototype.SetActi vi tyStatus=function(pStatus, pDateMethodInv){
    Siebel JS. Log(' Service Method SetActi vi tyStatus. . . ');
    var strstatval ue;
    var pi ckName;
```

```

var pickListDef;
var pModel;
var pBusComp;
pModel = SiebelApp.S_App.GetModel();
var pBusObj = pModel.GetBusObject("boName");
pBusComp = pBusObj.GetBusComp("bcName");
SiebelApp.S_App.GetActiveView().GetActiveApplet().GetControl("Status").GetPickListDef();
$.callBack(this, function(retObj){
    pickName = retObj.retVal;
    $.callBack(this, function(retObj){
        pickListDef=retObj.retVal;
        pModel=SiebelApp.S_App.Model;
        pModel.GetLovNameVal("Acknowledged", pickListDef.LOVType);
        $.callBack(this, function(retObj){
            strstatval=retObj.retVal;
            pBusComp.ActivateField("Status");
            $.callBack(this, function(retObj){
                pBusComp.SetFieldVal("Status", strstatval, true);
                $.callBack(this, function(retObj){
                    });
                });
            });
        });
        pBusComp.ActivateField("Status");
        $.callBack(this, function(retObj){
            pBusComp.SetFieldVal("Status", strstatval, true);
            $.callBack(this, function(retObj){
                });
            });
        });
        if(pDateMethodInv!="")//Todo - Refine this condition for uninitialized/defined
        or remove this condition
        {
            var now=new Date();
            if(pDateMethodInv == "ArrivedStatus")
            {
                pBusComp.SetFieldVal("Started", now, true);
                $.callBack(this, function(retObj){
                    pBusComp.SetFieldVal("Done", "", true);
                    $.callBack(this, function(retObj){
                        });
                    });
                });
            }
            else if(pDateMethodInv=="FinishedStatus")
            {
                pBusComp.SetFieldVal("Done", now, true);
                $.callBack(this, function(retObj){
                    pBusComp.SetFieldVal("Percent Complete", "100%", true);
                    $.callBack(this, function(retObj){
                        });
                    });
                });
            }
        }
    }
}

```

```
    }  
    pBusComp.WriteRecord();  
};
```

For information about the methods that this code uses, see [“callback Method” on page 419](#) and [“SetFieldValue Method” on page 404](#), and [“WriteRecord Method” on page 407](#).

5 Test your modifications:

- a** Log in to the disconnected client.
- b** Update the status of an activity.

Make sure Siebel Open UI displays the correct status activity. For example, if you set the status to Acknowledged, then make sure Siebel Open UI allows you to choose the EnRoute status and disables all other values.

Methods You Can Use to Customize Siebel Mobile Disconnected

This topic describes the methods that exist in the Application Programming Interface that you can use to customize Siebel Mobile Disconnected in Siebel Open UI. It includes the following information:

- [“Methods You Can Use in the Applet Class” on page 387](#)
- [“Methods You Can Use in the Business Component Class” on page 390](#)
- [“Methods You Can Use in the Business Object Class” on page 408](#)
- [“Methods You Can Use in the Business Service Class” on page 410](#)
- [“Methods You Can Use in the Application Class” on page 413](#)
- [“Methods You Can Use in the Model Class” on page 417](#)
- [“Methods You Can Use in the Service Model Class” on page 418](#)
- [“Methods You Can Use in Offline Classes” on page 418](#)
- [“Other Methods You Can Use with Siebel Mobile Disconnected” on page 420](#)

You can configure Siebel Open UI to override or customize some of the methods that this topic describes. For more information about how to customize or override a method, see [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#).

Methods You Can Use in the Applet Class

This topic describes methods that you can use that reside in the Applet class. It includes the following information:

- [“BusComp Method for Applets” on page 388](#)
- [“BusObject Method for Applets” on page 388](#)
- [“CanInvokeMethod Method” on page 388](#)

- ["InvokeMethod Method for Applets" on page 389](#)
- ["Name Method for Applets" on page 389](#)

BusComp Method for Applets

The BusComp method returns the business component that the applet references. It uses the following syntax:

```
Applet.BusComp()
```

For example, the following code gets the metadata for the business component that the active applet references:

```
SiebelApp.S_App.FindApplet(appletName).BusComp();
```

Each applet references a business component. If you configure Siebel Open UI to call BusComp on an applet, then it returns the business component that this applet references.

The BusComp method includes no arguments.

For information about using BusComp in the context of a business object, see ["GetBusComp Method for Business Objects" on page 409](#).

BusObject Method for Applets

The BusObject method returns the business object that the business component references. It uses the following syntax:

```
Applet.BusObject()
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusObject();
```

The BusObject method includes no arguments.

CanInvokeMethod Method

The CanInvokeMethod method determines whether or not Siebel Open UI can call a method. It returns the following properties. If you use CanInvokeMethod, then you must configure it so that it returns these properties:

- **Invoked.** This property returns one of the following values:
 - **true.** Siebel Open UI examined the method.
 - **false.** Siebel Open UI did not examine the method.
- **RetVal.** This property returns one of the following values:
 - **true.** Siebel Open UI can call the method.
 - **false.** Siebel Open UI cannot call the method.

The CanInvokeMethod method uses the following syntax:

```
Applet.CanInvokeMethod(methodName)
```

where:

- *methodName* is a string that contains the name of the method that CanInvokeMethod examines. CanInvokeMethod gets this string as a property that resides in an input property set.

For examples that use CanInvokeMethod, see the following topics:

- [“Using Custom JavaScript Methods” on page 364](#)
- [“Using Custom Siebel Business Services” on page 367](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Return Parts” on page 378](#)

InvokeMethod Method for Applets

The InvokeMethod method calls a method. If you use InvokeMethod, then you must configure it so that it returns a property set that includes one of the following values:

- **true.** Siebel Open UI called the method.
- **false.** Siebel Open UI did not call the method.

It uses the following syntax:

```
Applet.InvokeMethod(methodName);
```

where:

- *methodName* is the value of an input property that identifies the method that InvokeMethod calls.

For example, InvokeMethod in the following code calls the method that the value of the svcMthdName variable contains:

```
Applet.InvokeMethod(svcMthdName);
```

For examples that use InvokeMethod, see [“Using Custom JavaScript Methods” on page 364](#) and [“Allowing Users to Commit Part Tracker Records” on page 376](#).

Name Method for Applets

The Name method for an applet returns the name of an applet. It uses the following syntax:

```
Applet.Name()
```

For example:

```
SiebelApp.S_App.GetActiveView().GetActiveApplet().Name();
```

The Name method includes no arguments.

Methods You Can Use in the Business Component Class

This topic describes methods that you can use that reside in the Business Component class. It includes the following information:

- [“ActivateField Method” on page 391](#)
- [“ActivateMultipleFields Method” on page 392](#)
- [“Associate Method” on page 393](#)
- [“ClearToQuery Method” on page 393](#)
- [“CountRecords Method” on page 394](#)
- [“DeactivateFields Method” on page 395](#)
- [“DeleteRecord Method” on page 395](#)
- [“ExecuteQuery Method” on page 395](#)
- [“FirstRecord Method” on page 396](#)
- [“GetAssocBusComp Method” on page 396](#)
- [“GetFieldValue Method” on page 397](#)
- [“GetLinkDef Method” on page 398](#)
- [“GetLastErrCode Method for Business Components” on page 398](#)
- [“GetLastErrText Method for Business Components” on page 399](#)
- [“GetMultipleFieldValues Method” on page 399](#)
- [“GetPicklistBusComp Method” on page 400](#)
- [“GetSearchExpr Method” on page 401](#)
- [“GetSearchSpec Method” on page 401](#)
- [“GetUserProperty Method” on page 402](#)
- [“GetViewMode Method” on page 402](#)
- [“InvokeMethod for Business Components” on page 402](#)
- [“Name Method for Business Components” on page 402](#)
- [“NextRecord Method” on page 403](#)
- [“ParentBusComp Method” on page 403](#)
- [“Pick Method” on page 403](#)
- [“RefreshBusComp Method” on page 403](#)
- [“RefreshRecord Method” on page 404](#)
- [“SetFieldValue Method” on page 404](#)
- [“SetMultipleFieldValues Method” on page 404](#)
- [“SetSearchSpec Method” on page 405](#)

- ["SetViewMode Method" on page 406](#)
- ["UndoRecord Method" on page 406](#)
- ["UpdateRecord Method" on page 407](#)
- ["WriteRecord Method" on page 407](#)

ActivateField Method

The ActivateField method activates a business component field. It returns nothing. It uses the following syntax:

```
this.s.ActivateField(field_name);  
bc.ActivateField("field_name");// calling from another JavaScript file
```

where:

- *field_name* identifies the name of a business component field.

A field is inactive except in the following situations, by default:

- The field is a system field, such as Id, Created, Created By, Updated, or Updated By.
- The Force Active property of the field is TRUE.
- The Link Specification property of the field is TRUE.
- An active applet includes the field, and this applet references a business component that is active.
- The field resides in an active list applet, and the Show In List property of the list column that displays this field in the applet is TRUE.

Note the following:

- Siebel CRM calls the ActivateField method on the field, and then runs the ExecuteQuery method.
- If Siebel CRM calls the ActivateField method after it calls the ExecuteQuery method, then the ActivateField method deletes the query context.
- The ActivateField method causes Siebel CRM to include the field in the SQL statement that the ExecuteQuery method starts. If Siebel CRM activates a field, and if a statement in the GetFieldValue method or the SetFieldValue method references the field before Siebel CRM performs a statement from the ExecuteQuery method, then the activation has no effect.

Example

The following example uses the ActivateField method to activate the Login Name field that resides in the Contact business component:

```
var model = Siebel.App.S_App.GetModel();  
var boContact = model.GetBusObject("Contact");  
var bcContact = boContact.GetBusComp("Contact");  
bcContact.ClearToQuery();  
bcContact.ActivateField("Login Name");  
var sLoginName = "SPORTER";  
bcContact.SetSearchSpec("Login Name", sLoginName);
```

```
bcContact.ExecuteQuery();
$.callback(this, function () {
  if (!retObj.err) {
    model.ReleaseBO(bcContact);
  }
}
```

ActivateMultipleFields Method

The ActivateMultipleFields method activates more than one field. It returns nothing. It uses the following syntax:

```
BusComp.ActivateMultipleFields(Siebel PropertySet);
```

where:

- Siebel PropertySet is a property set that identifies a collection of properties. These properties identify the fields that Siebel CRM must activate.

Example 1

The following example uses the ActivateMultipleFields method to activate all the fields that the property set contains, including the Account Products, Agreement Name, Project Name, Description, and Name fields:

```
var ps = Siebel App. S_App.NewPropertySet();
ps.setProperty("Account Products", "");
ps.setProperty("Agreement Name", "");
ps.setProperty("Project Name", "");
ps.setProperty("Description", "");
ps.setProperty("Name", "");
BusComp.ActivateMultipleFields(ps);
```

Example 2

The following example in Siebel eScript queries the Contact business component and returns the First Name and Last Name of the first contact that it finds:

```
var model = Siebel App. S_App.GetModel();
var ContactBC = model.GetBusObject("Contact");
var ContactBC = bcContact.GetBusComp("Contact");
if (ContactBC)
{
  var fieldsPS = Siebel App. S_App.NewPropertySet();
  var valuesPS = Siebel App. S_App.NewPropertySet();
  fieldsPS.SetProperty("Last Name", "");
  fieldsPS.SetProperty("First Name", "");
  ContactBC.ActivateMultipleFields(fieldsPS);
  ContactBC.ClearToQuery();
  ContactBC.ExecuteQuery();
  $.callback(this, function () {
    if (!retObj.err) {
      ContactBC.FirstRecord();
      $.callback(this, function () {
```



```
if (!retObj.err) {  
    ContactBC .GetMultipleFieldValues(fieldsPS, valuesPS);  
    var slName = valuesPS.GetProperty("Last Name");  
    var sfName = valuesPS.GetProperty("First Name");  
}  
}  
}
```

Associate Method

The Associate method adds an association between the active record that resides in the child association business component and the parent business component. You can customize or override this method. It returns the retObj object with err set to one of the following values:

- **true.** The Associate method successfully added the record.
- **false.** The Associate method did not successfully add the record.

It uses the following syntax:

```
BusComp. Associate()
```

where:

- BusComp identifies an instance of the child business component.

For example:

```
Siebel App. S_App. FindApplet(appletName). BusComp(). Associate();
```

It includes no arguments.

An *association business component* is a type of business component that includes an intertable. For more information, see [“GetAssocBusComp Method” on page 396](#).

ClearToQuery Method

The ClearToQuery method clears the current query. It returns nothing. It uses the following syntax:

```
BusComp. ClearToQuery();
```

It includes no arguments.

Note the following:

- The ClearToQuery method does not clear the sort specification that Siebel Open UI defines in the Sort Specification property of a business component.
- You must use the ActivateField method to activate a field before you can use the ClearToQuery method. For more information see [“ActivateField Method” on page 391](#).

- Any search specifications and sort specifications that Siebel Open UI sends to a business component are cumulative. The business component performs an AND operation for the queries that accumulate since the last time Siebel CRM performed the `ClearToQuery` method. An exception to this configuration occurs if Siebel Open UI adds a new search specification to a field, and if this field already includes a search specification. In this situation, the new search specification replaces the old search specification.

Example

The following example uses the `ClearToQuery` method:

```
var model = Siebel App. S_App. GetModel ();
var oEmpBusObj = model . GetBusObj ect("Empl oyee");
var oEmpBusComp = oEmpBusObj . GetBusComp("Empl oyee ");
var sLogi nName;
oEmpBusComp. Cl earToQuery();
oEmpBusComp. SetSearchSpec("Logi n Name", sLogi nName);
oEmpBusComp. ExecuteQuery();
```

For another example usage of the `ClearToQuery` method, see [“CountRecords Method” on page 394](#).

CountRecords Method

The `CountRecords` method returns the number of records that a business component contains according to the search specification and query specification that Siebel Open UI runs on this business component. It uses the following syntax:

```
BusComp. CountRecords();
```

It includes no arguments.

Example

The following example uses the `CountRecords` method:

```
var model = Siebel App. S_App. GetModel ();
var bo = model . GetBusObj ect("Opportuni ty ");
var bc = bo. GetBusComp("Opportuni ty");
i f (bc)
{
    bc . Cl earToQuery();
    bc . SetSearchSpec ("Name", "A");
    bc . ExecuteQuery();
    $. call back(this, function () {
        i f (!retObj .err) {
            var count = bc . CountRecords();
            $. setReturnVal ue({err: fal se, retVal : count});
        }
    })
}
```

For more information, see [“ClearToQuery Method” on page 393](#).

DeactivateFields Method

The DeactivateFields method deactivates fields from the SQL query statement of a business component. It deactivates fields that are currently active. DeactivateFields applies this behavior except in the following situations:

- The Force Active property is TRUE.
- A link requires the field to remain active.
- A business component class requires the field to remain active.

The DeactivateFields method returns nothing.

It uses the following syntax:

```
BusComp.DeactivateFields()
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().DeactivateFields();
```

It includes no arguments.

You must use the ActivateField method to activate a field before you configure Siebel Open UI to perform a query for a business component. After Siebel Open UI deactivates a field, you must configure it to query the business component again or the Siebel application fails.

DeleteRecord Method

The DeleteRecord method deletes the current record from the local database. It returns one of the following values:

- **error:false.** DeleteRecord deleted the record.
- **error:true.** DeleteRecord did not delete the record.

It uses the following syntax:

```
buscomp.DeleteRecord(psiInputArgs);
```

ExecuteQuery Method

The ExecuteQuery method runs a query according to the current value of the Search Specification property, the current value of the Sort Specification property, or according to both of these properties. The business component contains these properties. ExecuteQuery runs this query on the local database. It returns one of the following values:

- If an error occurs, then it returns err with an error message. For example:

```
{err: "Error Message", retVal: ""}
```

- If an error does not occur, then it returns an empty err message. For example:

```
{err: "", retVal: ""}
```

It uses the following syntax:

```
busComp. ExecuteQuery();
```

where:

- busComp identifies the business component that ExecuteQuery uses to get the search specification or sort specification. You can use busComp as a literal or a variable. For more information, see [“How This Book Indicates Code That You Can Use as a Variable and Literal” on page 31](#).

FirstRecord Method

The FirstRecord method moves the record pointer to the first record in a business component, making this record the current record. It uses the following syntax:

```
BusComp. FirstRecord();
```

For example:

```
Siebel App. S_App. FindApplet(appletName). BusComp(). FirstRecord();
```

GetAssocBusComp Method

The GetAssocBusComp method returns an instance of the association business component. It uses the following syntax:

```
BusComp. GetAssocBusComp();
```

It includes no arguments.

For more information, see [“Associate Method” on page 393](#).

You can use an association business component to manipulate an association. You can use the GetAssocBusComp method and the Associate method only with a many-to-many relationship that uses an intersection table. For example, with accounts and contacts.

Note the following:

- To associate a new record, you add it to the child business component.
- To add a record, you use the GetAssocBusComp method and the Associate method.

If a many-to-many link exists, and if Siebel CRM defines an association applet for the child applet, then you can use the GetAssocBusComp method with the child business component of a parent-child view.

Example of Using the GetAssocBusComp Method

The following example associates a contact that includes the ContactID Id with an account that includes the AccountId Id:

```
var Model = Siebel App. S_App. GetModel ();  
var account B0 = Model . GetBusObj ("Account");  
var accountBC = accountB0. GetBusComp("Account");  
var contactBC = accountB0. GetBusComp("Contact");  
accountBC. SetSearchSpec("Id", [AccountId]);
```

```
accountBC. ExecuteQuery ();
$.call back(this, function(){
accountBC. FirstRecord(); // positions on the account record
$.call back(this, function(){
    contactBC. ExecuteQuery ();
    $.call back(this, function(){
        contactBC. FirstRecord();
        $.call back(this, function(){
            var assocBC = contactBC. GetAssocBusComp();
            assocBC. SetSearchSpec("Id", [ContactID]);
            assocBC. ExecuteQuery ();
            $.call back(this, function(){

                assocBC. FirstRecord(); // positions on the contactbc
                $.call back(this, function(){
                    contactBC. Associate() // adds the association
                })
            })
        })
    });
});
});
```

GetFieldValue Method

The GetFieldValue method returns the value of a field for the current record or for the record object that Siebel Open UI examines. It uses the following syntax:

```
Buscomp. GetFi el dVal ue(" fi el d_name", pRecord)
```

where:

- *field_name* is a string that contains the name of a field. Siebel Open UI returns the value that this field contains.
- pRecord is an optional argument that returns the entire record that Siebel Open UI examines. If you do not specify pRecord, or if it is empty, then GetFieldValue returns only a value in *field_name* of the active record.

For example, the following code returns the value of the Account Name field from the current record of the business component:

```
Buscomp. GetFi el dVal ue("Account Name")
```

For another example, the following code returns the field value of the Account Name field. A business component can include more than one record, but only one of these records is the active record. You can use pRecord to get the value of a field from a record that is not the active record:

```
Buscomp. GetFi el dVal ue("Account Name", recordObj ect)
```

The GetFieldValue method returns an object that includes an error code and a return value. For more information, see [“Configuring Error Messages for Disconnected Clients” on page 369](#) and [“SetErrorMsg Method” on page 420](#).

For more examples that use the GetFieldValue method, see the following topics:

- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Commit Part Tracker Records” on page 376](#)
- [“Allowing Users to Return Parts” on page 378](#)

You can configure Siebel Open UI to override the `GetFieldValue` method.

GetLinkDef Method

The `GetLinkDef` method returns the link definition of the child business component. This business component is the child in the parent and child relationship of a link. It returns this definition after Siebel Open UI processes data for the child business component. This definition includes values for the following properties:

- Name
- RecordNum
- childBusCompName
- destFieldName
- interChildColName
- interParentColName
- interTableName
- parentBusCompName
- primelIdFieldName
- searchSpec
- sortSpec
- srcFieldName
- NoDelete
- NoInsert
- NointerDelete
- NoUpdate
- SrcFieldValue

If the value of a property is empty, then `GetLinkDef` does not return this property in the return object.

The `GetLinkDef` method uses the following syntax:

```
Linkdef = busComp.GetLinkDef();  
var sourceFieldName = Linkdef.srcFieldName;
```

GetLastErrCode Method for Business Components

The `GetLastErrCode` method returns the error code for the most recent error that the disconnected client logged. It uses the following syntax:

```
BusComp.GetLastErrorCode()
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().GetLastErrorCode();
```

This method includes no arguments.

The error code that this method returns is a short integer. An error code of 0 (zero) indicates no error occurred.

GetLastErrorText Method for Business Components

The GetLastErrorText method returns a string that contains the text message for the most recent error that the disconnected client logged. It uses the following syntax:

```
BusComp.GetLastErrorText()
```

For example:

```
ActiveBusObject().GetLastErrorText();
```

This method includes no arguments.

GetMultipleFieldValues Method

The GetMultipleFieldValues method returns a value for each field that a property set specifies. It uses the following syntax:

```
BusComp.GetMultipleFieldValues(fieldNamePropSet, fieldValuePropSet)
```

where:

- `fieldNamePropSet` is a property set that identifies a collection of fields.
- `fieldValuePropSet` is a property set that includes values for the fields that the `fieldNamePropSet` argument specifies.

If an error occurs, then GetMultipleFieldValues returns `err` with an error message. For example:

```
{err: "Error Message", retVal: ""}
```

If an error does not occur, then GetMultipleFieldValues returns an empty `err` message. For example:

```
{err: "", retVal: ""}
```

You cannot use the same instance of a property set for the `fieldNamePropSet` argument and for the `fieldValuesPropSet` argument.

Example of Using the GetMultipleFieldValues Method

The following example uses the GetMultipleFieldValues method:

```
var oPSDR_Header = SiebelApp.S_App.NewPropertySet();  
// Cannot use the same property set in GetMultipleFieldValues, must use a different  
// one for the values. The process will not error, but Siebel Open UI will not place
```

```
// the values in the property set.
var IPS_values = SiebelApp.S_App.NewPropertySet();
oPsDR_Header.SetProperty("Last Name", "");
oPsDR_Header.SetProperty("First Name", "");
oPsDR_Header.SetProperty("Middle Name", "");
var model = SiebelApp.S_App.GetModel();
var boContact = model.GetBusObject("Contact");
var bcContact = boContact.GetBusComp("Contact");
bcContact.ActivateMultipleFields(oPsDR_Header);
bcContact.SetSearchSpec("Last Name", "Mead*");
ExecuteQuery();
$.callback(this, function(){
    FirstRecord();
    $.callback(this, function(){
        // Use a different property set for the values. If you use the same one
        // for arguments you get no values back.
        GetMultipleFieldValues(oPsDR_Header, IPS_values);
        // Get the value from the output property set.
        $.callback(this, function(){
            SiebelJS.Log("Full Name is " + IPS_values.GetProperty("First Name") +
            IPS_values.GetProperty("Middle Name") + IPS_values.GetProperty("Last Name"));
        });
    });
});
```

GetPicklistBusComp Method

The GetPicklistBusComp method returns a pick business component that Siebel CRM associates with a field that resides in the current business component. If no picklist is associated with this field, then this method returns an error. It uses the following syntax:

```
BusComp.GetPicklistBusComp(FieldName)
```

You can use the GetPicklistBusComp method to manipulate a picklist, and you can use the name of the pick business component that the GetPicklistBusComp method returns.

How Siebel Open UI Uses the GetPickListBusComp Method With Constrained Picklists

If Siebel CRM uses the GetPickListBusComp method or the Pick method to pick a record that resides in a constrained picklist, then the constraint is active. The pick business component that these methods return contains only the records that meet the constraint.

Configuring Siebel Open UI to Pick a Value from a Picklist

This topic describes how to configure Siebel Open UI to pick a value from a picklist.

To configure Siebel Open UI to pick a value from a picklist

- 1 Use a JavaScript editor to open the JavaScript file that you must modify. This file resides on the client.

2 Add code that uses the Pick method to pick the value.

For example, add the following code to the method that Siebel Open UI uses to register the service:

```
this.GetFieldValue("City")
$.callback(this, function(retObj){
  if(retObj.retVal === "San Mateo")
  {
    var oBCPick = this.GetPicklistBusComp("State");
    oBCPick.SetSearchSpec("Value", "CA");
    oBCPick.ExecuteQuery(ForwardOnly);
    $.callback(this, function(){
      oBCPick.FirstRecord();
      $.callback(this, function(){
        if(oBCPick.CheckActiveRow()){
          oBCPick.Pick();
        }
      });
    });
  }
});
}
```

This code configures Siebel Open UI to use the GetPicklistBusComp method to create an instance of the picklist business component. For more information, see ["Pick Method" on page 403](#).

GetSearchExpr Method

The GetSearchExpr method returns a string that contains the current search expression that Siebel Open UI defines for a business component. The following search expression is an example of a string that GetSearchExpr might return:

```
[Revenue] > 10000 AND [Probability] > .5
```

The GetSearchExpr method uses the following syntax:

```
BusComp.GetSearchExpr();
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().GetSearchExpr();
```

The GetSearchExpr method includes no arguments.

If an instance of the business component does not exist, then the GetSearchExpr method returns nothing.

GetSearchSpec Method

The GetSearchSpec method returns a string that contains the search specification that Siebel Open UI defines for a business component field in. For example, it might return the following search specification:

```
> 10000
```

The GetSearchSpec method uses the following syntax:

```
BusComp. GetSearchSpec(Fiel dName);
```

For example:

```
Si ebel App. S_App. Fi ndAppl et (appl etName). BusComp(). GetSearchSpec (Fi el dName);
```

GetUserProperty Method

The GetUserProperty method gets the value of a business component user property. It uses the following syntax:

```
BusComp. GetUserProperty(business_component_user_property)
```

where:

- *business_component_user_property* is a string that identifies the name of a business component user property.

For example, the following code gets the value of the Deep Copy business component user property:

```
Si ebel App. S_App. Fi ndAppl et (appl etName). BusComp(). GetUserProperty ("Deep Copy");
```

GetViewMode Method

The GetViewMode method returns a Siebel ViewMode constant or the corresponding integer value for this constant. This constant identifies the current visibility mode of a business component. This mode determines the records that a query returns according to the visibility rules.

The GetViewMode method uses the following syntax:

```
BusComp. GetVi ewMode()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. Fi ndAppl et (appl etName). BusComp(). GetVi ewMode();
```

InvokeMethod for Business Components

The InvokeMethod method that you can use with business components works the same as the InvokeMethod method that you can use with applets. For more information about the InvokeMethod method that you can use with applets, see [“InvokeMethod Method for Applets” on page 389](#).

Name Method for Business Components

The Name method returns the name of a business component. It uses the following syntax:

```
Si ebel App. S_App. Fi ndAppl et (appl etName). BusComp()
```

It includes no arguments.

NextRecord Method

The NextRecord method moves the record pointer to the next record that the business component contains, making this next record the current record. It adds the next record that the current search specification and sort specification identifies, and then sets the active row to this record. It adds this record to the current set of records. It does this work only if the current set of records does not already contain this next record. It returns this next record. It uses the following syntax:

```
BusComp.NextRecord()
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().NextRecord();
```

It includes no arguments.

ParentBusComp Method

The ParentBusComp method returns the parent business component of a business component. It uses the following syntax:

```
BusComp.ParentBusComp()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().ParentBuscomp()
```

Pick Method

The Pick method places the currently chosen record that resides in a pick business component into the appropriate fields of the parent business component. It uses the following syntax:

```
BusComp.Pick()
```

The Pick method includes no arguments.

You cannot use the Pick method to modify the record in a picklist field that is read-only.

For usage information, see [“Configuring Siebel Open UI to Pick a Value from a Picklist” on page 400](#).

For more information about pick business component, see *Configuring Siebel Business Applications*.

RefreshBusComp Method

The RefreshBusComp method runs the current query again for a business component and makes the record that was previously active the active record. The user can view the updated view, but the same record remains highlighted in the same position in the list applet. This method returns nothing.

It uses the following syntax:

```
BusComp.InvokeMethod("RefreshBusComp")
```

For example:

```
"buscomp. InvokeMethod("RefreshBusComp") $.callback(this, function (retObj) {  
  if (!retObj.err) {}});"
```

It includes no arguments.

RefreshRecord Method

The RefreshRecord method updates the currently highlighted record and the business component fields in the Siebel client. It positions the cursor on the highlighted record. It does not update other records that are currently available in the client. This method returns nothing.

It uses the following syntax:

```
BusComp. InvokeMethod("RefreshRecord ")
```

For example:

```
"buscomp. InvokeMethod("RefreshRecord") $.callback(this, function (retObj) {  
  if (!retObj.err) {}});"
```

It includes no arguments.

SetFieldValue Method

The SetFieldValue method sets a field value in a record. It returns one of the following values depending on whether it successfully set the field value:

- **Successfully set the field value.** Returns an empty error code.
- **Did not successfully set the field value.** Returns an error code.

It uses following syntax.

```
SetFieldVal ue(fieldName, fieldValue);
```

where:

- *fieldName* is a string that contains the name of the field that SetFieldValue updates.
- *fieldValue* is a string that contains the value that SetFieldValue uses to update the field.

For examples that use the SetFieldValue method, see the following topics:

- [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Commit Part Tracker Records” on page 376](#)
- [“Allowing Users to Return Parts” on page 378](#)
- [“Allowing Users to Set the Activity Status” on page 384](#)

SetMultipleFieldValues Method

The SetMultipleFieldValues method sets new values in the fields of the current record of a business component. It uses the following syntax:

```
BusComp.SetMultipleFieldValues (oPropertySet)
```

The `FieldName` argument that the property set contains must match the field name that Siebel Tools displays. This match must be exact, including upper and lower case characters.

In the following example, the `FieldName` is `Name` and the `FieldValue` is `Acme`:

```
oPropertySet.SetProperty ("Name", "Acme")
```

Note the following:

- If an error occurs in the values of any of fields that the property set specifies, then Siebel Open UI stops the process it is currently running.
- You can use the `SetMultipleFieldValues` method only on a field that is active.
- You must not use the `SetMultipleFieldValues` method on a field that uses a picklist.

Example

The following example in Siebel eScript uses the `SetMultipleFieldValues` method to set the values for all fields that the property set identifies, including the `Name`, `Account`, and `Sales Stage`:

```
var model = SiebelApp.S_App.GetModel();
var bo = model.GetBusObj("Opportunity");
var bc = bo.GetBusComp("Opportunity");
var ps = SiebelApp.S_App.NewPropertySet();
ps.SetProperty("Name", "Call Center Opportunity");
ps.SetProperty("Account", "Marriott International");
ps.SetProperty("Sales Stage", "2-Qualified");
bc.ActivateMultipleFields(ps);
bc.NewRecord();
$.callBack(this, function(){
    bc.SetMultipleFieldValues(ps);
    $.callBack(this, function(){
        ps = null;
        bc.WriteRecord();
    });
});
```

SetSearchSpec Method

The `SetSearchSpec` method sets the search specification for a business component. It returns nothing. It uses the following syntax:

```
BusComp.SetSearchSpec(FieldName, searchSpec);
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().SetSearchSpec("Id", strCallId);
```

where:

- *FieldName* is a string that identifies the name of the field where Siebel Open UI sets the search specification.

- *searchSpec* is a string that contains the search specification.

You must configure Siebel Open UI to call the `SetSearchSpec` method before it calls the `ExecuteQuery` method. To avoid an unexpected compound search specification on a business component, it is recommended that you configure Siebel Open UI to call the `ClearToQuery` method before it calls the `SetSearchSpec` method.

SetViewMode Method

The `SetViewMode` method sets the visibility type for a business component. It returns nothing. It uses the following syntax:

```
BusComp.SetViewMode(i nMode);
```

where:

- *i nMode* identifies the view mode. It contains one of the following integers:
 - **0.** Sales Representative.
 - **1.** Manager.
 - **2.** Personal.
 - **3.** All.
 - **4.** None.
 - **5.** Organization.
 - **6.** Contact.

For example:

```
Siebel App. S_App. FindApplet(appletName). BusComp(). SetViewMode(i nMode);
```

UndoRecord Method

The `UndoRecord` method reverses any unsaved modifications that the user makes on a record. This includes reversing unsaved modifications to fields, and deleting an active record that is not saved. It returns one of the following values:

- **true.** `UndoRecord` successfully deleted the record.
- **false.** `UndoRecord` did not successfully delete the record.

It uses the following syntax:

```
BusComp.UndoRecord();
```

It includes no arguments.

For example:

```
Siebel App. S_App. FindApplet(appletName). BusComp(). UndoRecord();
```

You can use the `UndoRecord` method in the following ways:

- To delete a new record. Use it after Siebel CRM calls the NewRecord method and before it saves the new record to the Siebel database.
- To reverse modifications that the user makes to field values. Use it before Siebel CRM uses the WriteRecord method to save these changes, or before the user steps off the record.

UpdateRecord Method

The UpdateRecord method places the current record in the commit pending state so that Siebel Open UI can modify it. It returns the retObj object with retVal set to one of the following values:

- **true.** The UpdateRecord method successfully placed the current record in the commit pending state.
- **false.** The UpdateRecord method did not successfully place the current record in the commit pending state.

It uses the following syntax:

```
thi s.UpdateRecord();
```

where:

- **thi s** identifies a business component instance.

For example, the following code calls the CanUpdate method. If CanUpdate indicates that Siebel Open UI can update the active row, then this code places the current record in the commit pending state for the business component that **thi s** specifies:

```
thi s. UpdateRecord(fal se)
```

The UpdateRecord method can run in a Siebel Mobile disconnected client.

For more information, see [“CanUpdate Method” on page 439](#).

WriteRecord Method

The WriteRecord method writes any modifications that the user makes to the current record. If you use this method with:

- **A connected client.** WriteRecord writes these modifications to the Siebel Database that resides on the Siebel Server.
- **Siebel Mobile disconnected.** WriteRecord writes these modifications to the local database that resides on the client.

The WriteRecord method returns one of the following values:

- **error:false.** WriteRecord successfully wrote the modifications to the local database.
- **error:true.** WriteRecord did not successfully write the modifications to the local database.

The WriteRecord method uses the following syntax:

```
buscomp.wri terecord(bAddSyncQ)
```

where:

- `bAddSyncQ` is an optional argument that specifies to synchronize the modification that `WriteRecord` makes to the Siebel Server. You can set this argument to one of the following values:
 - **true**. Siebel Open UI synchronizes the modification. This is the default setting.
 - **false**. Siebel Open UI does not synchronize the modification.

For examples that use the `WriteRecord` method, see the following topics:

- [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#)
- [“Customizing Predefined Business Components” on page 361](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Commit Part Tracker Records” on page 376](#)

Example

You must first configure Siebel Open UI to create new records and set values for fields. You can then use the following code to call the `WriteRecord` method to save the new record to the offline database:

```
var model = Siebel App. S_App. GetModel ();
var bo = model .GetBusObject("Opportunity ");
var bc = bo.GetBusComp("Opportunity");
var strDEANumber = 9089;
var strDEAExpDate = 02/12/2013;
bc.SetFieldVal ue("DEA#", strDEANumber);
$.call back(this, function () {
if (!retObj.err) {
    bc.SetFieldVal ue("DEA Expi ry Date", strDEAExpDate);
    $.call back(this, function () {
if (!retObj.err) {
    bc.SetFieldVal ue("DEA Expi ry Date", strDEAExpDate);
    $.call back(this, function () {
if (!retObj.err) {
    bc.Wri teRecord();
}
}
}
}
```

Methods You Can Use in the Business Object Class

This topic describes methods that you can use that reside in the Business Object class. It includes the following information:

- [“GetBusComp Method for Business Objects” on page 409](#)
- [“GetLastErrCode Method for Business Objects” on page 409](#)
- [“GetLastErrText Method for Business Objects” on page 409](#)
- [“Name Method for Business Objects” on page 410](#)

GetBusComp Method for Business Objects

The GetBusComp method returns the business component instance that a business object references. It uses the following syntax:

```
Siebel App. S_App. Model . GetBusObj (business_object) . GetBusComp(business_component)
```

where:

- *business_object* identifies the name of a business object.
- *business_component* identifies the name of a business component.

Each view references a business object, and each business object references one or more business components. If you configure Siebel Open UI to call GetBusComp in the context of a business object, then you must do the following:

- use the *business_object* argument to specify the name of the business object that the view references.
- use the *business_component* argument to specify the name of a business component that the business object references.

For example, the following code gets the business component instance for the Order Entry - Orders business component that the Service Request business object references:

```
Siebel App. S_App. Model . GetBusObj ("ServiceRequest") . GetBusComp("Order Entry - Orders")
```

For information about using BusComp in the context of an applet, see [“BusComp Method for Applets” on page 388](#). For more information about views, business objects, and business components, and how they reference each other, see *Configuring Siebel Business Applications*.

GetLastErrCode Method for Business Objects

The GetLastErrCode method returns the error code for the most recent error that the disconnected client logged. It uses the following syntax:

```
BusObj . GetLastErrCode()
```

For example:

```
ActiveBusObject(). GetLastErrCode();
```

This method includes no arguments.

The error code that this method returns is a short integer. An error code of 0 (zero) indicates no error occurred.

GetLastErrText Method for Business Objects

The GetLastErrText method returns a string that contains the text message for the most recent error that the disconnected client logged. It uses the following syntax:

```
BusObj . GetLastErrText()
```

For example:

```
ActiveBusObject().GetLastErrText();
```

This method includes no arguments.

Name Method for Business Objects

The Name method returns the name of a business object. It uses the following syntax:

```
BusObject.Name();
```

This method includes no arguments.

Methods You Can Use in the Business Service Class

This topic describes methods that you can use that reside in the Business Service class. It includes the following information:

- [“Invoke Method for Business Services” on page 410](#)
- [“ServiceRegistry Method” on page 411](#)

Invoke Method for Business Services

The Invoke method that you can use with a business service calls the CanInvokeMethod business service and the InvokeMethod business service. It returns a property set. It uses following syntax:

```
service.Invoke(method_name, psPropertySet);
```

where:

- *method_name* is a string that identifies the business service method that the Invoke method calls. The Invoke method also calls the following methods:
 - **CanInvokeMethod.** Determines whether or not Siebel Open UI can call the business service method that *method_name* identifies. Any custom business service file you create must include the CanInvokeMethod business service method.
 - **InvokeMethod.** Calls the business service method that *method_name* identifies. Any custom business service file you create must include the InvokeMethod business service method.

For more information about using these methods, see [“Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects” on page 361](#).

- *psPropertySet* is a property set that the Invoke method sends to the method that *method_name* identifies.

The following example calls the CanAddSample method of the LS Pharma Validation Service business service:

```
var service = SiebelApp.S_App.GetService("LS Pharma Validation Service");  
var outputSet = service.Invoke("CanAddSample", psPropertySet);
```

For an example that uses the Invoke method with a business service, see [“Using Custom Siebel Business Services” on page 367](#).

ServiceRegistry Method

The ServiceRegistry method registers a custom business service method that you define. You must use it any time that you configure Siebel Open UI to call a custom business service method. It returns one of the following values:

- **true.** Siebel Open UI successfully registered the method.
- **false.** Siebel Open UI did not successfully register the method.

It uses following syntax:

```
SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
```

where:

- `inputObj` is an object that specifies a set of properties, where each property specifies a name and a value. The number of properties varies according to object type. For a list of properties that you can use, see [“Properties You Must Include to Register Custom Business Services” on page 412](#). The `inputObj` argument uses the following syntax:

```
inputObj [oconsts.get("name")] = "value";
```

where:

- `name` specifies the property name
- `value` specifies the property value

For example, the following code specifies the DOUI REG_OBJ_NAME property with a value of Pharma Call Entry Mobile:

```
inputObj [oconsts.get("DOUI REG_OBJ_NAME")] = "Pharma Call Entry Mobile";
```

The following code specifies the property name:

```
oconsts.get("DOUI REG_OBJ_NAME")
```

Siebel Open UI registers a method for a custom service when it loads the script files that it uses for this custom service. This configuration makes sure that Siebel Open UI calls the ServiceRegistry method from the correct location in the code. To view this code in the context of a complete example, see [“Using Custom JavaScript Methods” on page 364](#).

Properties You Must Include to Register Custom Business Services

Table 24 describes the properties that you must include in the inputObj argument of the ServiceRegistry method when Siebel Open UI registers a custom business service. The local constants.js file defines each of these properties as a constant.

Table 24. Properties You Must Include to Register Custom Business Services

| Properties | Value |
|-------------------|--|
| DOUIREG_OBJ_NAME | The name of a custom business service. For example: LS Pharma Val i dati on Servi ce |
| DOUIREG_SRVC_NAME | The name of the JavaScript class that the custom business service references. For example: PharmaCal l Val i datorsvc |

Table 25 describes the properties you must include in the inputObj argument of the ServiceRegistry method when Siebel Open UI registers a custom business service that references a predefined applet or a predefined business component.

Table 25. Required Input Properties for Custom Business Services That Reference Predefined Applets or Business Components

| Property | Value |
|--------------------|---|
| DOUIREG_OBJ_NAME | Name of the applet or the business component that the custom business service method references. For example: Pharma Cal l Entry Mobi le |
| DOUI REG_OBJ_TYPE | Specifies that this business service method references an applet or a business component. You must use one of the following values: ■ Use DOUIREG_OBJ_TYPEAPPLET for an applet. ■ Use DOUIREG_OBJ_TYPEBUSCOMP for a business component. |
| DOUI REG_OBJ_MTHD | Name of the predefined business service method that you must customize. For example, WriteRecord. |
| DOUI REG_SRVC_NAME | The name of the JavaScript class that the Class property of the business service method references. For example: pharmacal l svc |

Table 25. Required Input Properties for Custom Business Services That Reference Predefined Applets or Business Components

| Property | Value |
|--------------------|--|
| DOUI REG_SRVC_MTDH | Name of the business service method that you customized. For example, WriteRecord. |
| DOUI REG_EXT_TYPE | <p>You can use one of the following values:</p> <ul style="list-style-type: none"> ■ DOUIREG_EXT_TYPEPRE. Siebel Open UI runs the custom business service method, and then runs the predefined business service method. You must configure Siebel Open UI to set the Invoked property to true after it processes DOUIREG_EXT_TYPEPRE so that it does not make any more calls to this method. ■ DOUIREG_EXT_TYPEPOST. Siebel Open UI runs the predefined business service method, and then runs the custom business service method. |

Methods You Can Use in the Application Class

This topic describes methods that you can use that reside in the Application class. It includes the following information:

- “ActiveBusObject Method” on page 413
- “ActiveViewName Method” on page 414
- “CurrencyCode Method” on page 414
- “FindApplet Method” on page 414
- “GetBusObject Method” on page 414
- “GetLastErrCode Method for Applications” on page 415
- “GetLastErrText Method for Applications” on page 415
- “GetService Method” on page 415
- “LoginId Method” on page 416
- “LoginName Method” on page 416
- “Name Method for Applications” on page 416
- “NewPropertySet Method” on page 416
- “PositionId Method” on page 417
- “PositionName Method” on page 417

ActiveBusObject Method

The ActiveBusObject method returns the business object that the active view references. It uses the following syntax:

```
Appl i cati on. Acti veBusObj ect()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. Acti veBusObj ect();
```

ActiveViewName Method

The ActiveViewName method returns the name of the active view. It uses the following syntax:

```
Appl i cati on. Acti veVi ewName()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. Acti veVi ewName();
```

CurrencyCode Method

The CurrencyCode method returns the currency code that Siebel CRM associates with the division of the user position. For example, USD for U.S. dollars, EUR for the euro, or JPY for the Japanese yen. It uses the following syntax:

```
Appl i cati on. CurrencyCode()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. CurrencyCode();
```

FindApplet Method

The FindApplet method returns the active applet. It uses the following syntax:

```
Appl i cati on. Fi ndAppl et (appl etName)
```

where:

- `appl etName` is a string that contains the name of the active applet.

For example, if the Contact List Applet is the current applet, then the `appl etName` variable in the following code returns the name of this applet as a string:

```
Si ebel App. S_App. Fi ndAppl et (appl etName);
```

GetBusObject Method

The GetBusObject method creates a new instance of a business object. It returns this new business object instance. It is not synchronous. It uses the following syntax:

```
Appl i cati on. GetBusObj ect (busi ness_obj ect_name)
```

where:

- *business_object_name* is a string that identifies the name of a business object

For example, the following code creates a new instance of the Opportunity business object:

```
Siebel App. S_App. GetBusinessObject(Opportunity);
```

GetLastErrCode Method for Applications

The `GetLastErrCode` method returns the error code for the most recent error that the disconnected client logged. It uses the following syntax:

```
Application.GetLastErrCode()
```

For example:

```
TheApplication().GetLastErrCode();
```

This method includes no arguments.

The error code that this method returns is a short integer. An error code of 0 (zero) indicates no error occurred.

GetLastErrText Method for Applications

The `GetLastErrText` method returns a string that contains the text message for the most recent error that the disconnected client logged. It uses the following syntax:

```
Application.GetLastErrText()
```

For example:

```
TheApplication().GetLastErrText();
```

This method includes no arguments.

GetService Method

The `GetService` method creates an instance of a business service object. It allows you to use the `Invoke` method to call this business service object. It uses the following syntax:

```
Siebel App. S_App. GetService("business_service_name");
```

where:

- *business_service_name* is a string that identifies the name of the business service that `GetService` uses to create the business service object. You must use the same name that you use when you register this business service. For more information about registering a business service, and for an example that uses the `GetService` method, see [“Using Custom Siebel Business Services” on page 367](#).

The following example creates a business service instance of the LS Pharma Validation Service business service:

```
var service = Siebel App. S_App. GetService("LS Pharma Validation Service");
```

LoginId Method

The LoginId method returns the login ID of the user who started the Siebel application. It uses the following syntax:

```
Application. LoginId()
```

It includes no arguments.

For example:

```
Siebel App. S_App. LoginId();
```

LoginName Method

The LoginName method returns the login name of the user who started the Siebel application. This login name is the name that the user enters in the login dialog box. It uses the following syntax:

```
Application. LoginName()
```

It includes no arguments.

For example:

```
Siebel App. S_App. LoginName();
```

Name Method for Applications

The Name method returns the name of the Siebel application. It uses the following syntax:

```
Application. Name()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Name();
```

NewPropertySet Method

The NewPropertySet method creates a new property set, and then returns this property set to the code that called this method. It uses the following syntax:

```
Application. NewPropertySet()
```

It includes no arguments.

For example:

```
Siebel App. S_App. NewPropertySet();
```


PositionId Method

The PositionId method returns the position ID of the user position. This position ID is the ROW_ID from the S_POSTN table. Siebel CRM sets this value when the Siebel application starts, by default. It uses the following syntax:

```
Appl i cati on. Posi ti onId()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. Posi ti onId();
```

PositionName Method

The PositionName method returns the name of the current user position. Siebel CRM sets this value when it starts the Siebel application, by default. It uses the following syntax:

```
Appl i cati on. Posi ti onName()
```

It includes no arguments.

For example:

```
Si ebel App. S_App. Posi ti onName();
```

Methods You Can Use in the Model Class

This topic describes methods that you can use that reside in the Model class.

GetLoginId Method

The GetLoginId method returns the login Id of the offline user who is currently logged in to the Siebel Mobile disconnected client. It uses the following syntax:

```
Var l ogi ni d = Si ebel App. S_App. Model . GetLogi nId();
```

ReleaseBO Method

The ReleaseBO method releases the current business object instance. It returns an instance of the current applet or current business component. It uses the following syntax:

```
Si ebel App. S_App. Model . Rel easeBO(obj B0);
```

where:

- obj B0 is a variable that identifies the business object instance that Siebel Open UI must release.

Methods You Can Use in the Service Model Class

This topic describes the method that you can use that resides in the Service Model class.

GetContext Method

The GetContext method gets the context that exists when a JavaScript service or a Siebel business service calls a method. It returns the current applet or business component depending on this context. It uses the following syntax:

```
serviceObj . GetContext()
```

You cannot configure Siebel Open UI to override this method.

Methods You Can Use in Offline Classes

This topic describes methods that you can use that reside in the offline classes. It includes the following information:

- ["setReturnValue Method" on page 418](#)
- ["callback Method" on page 419](#)
- ["eachAsyncOp Method" on page 419](#)
- ["SetErrorMsg Method" on page 420](#)

These methods reside in the OfflineAppMgr class, except for SetErrorMsg. It resides in the OfflineErrorObject class.

setReturnValue Method

The setReturnValue method sets the return value that Siebel Open UI sends to the method that calls the setReturnValue method. It uses the following syntax:

```
$. setReturnVal ue(return_val ue)
```

where:

- *return_value* identifies an object that includes the following information:
 - Error status of the code that Siebel Open UI called
 - retVal contains the return value of the code that Siebel Open UI called

For example:

```
$. setReturnVal ue({err: errCode, retVal : bRet})
```

where:

- errCode contains the error code that Siebel Open UI returns to the caller. For more information, see ["SetErrorMsg Method" on page 420](#).

If you do not use `setReturnValue`, then Siebel Open UI sends a `retObj` with `err` set to null and `retVal` set to empty, by default.

For examples that use the `setReturnValue` method, see [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#) and [“SetErrorMsg Method” on page 420](#).

callback Method

The callback method registers the done handler. it uses the following syntax:

```
$.callback (scope, done_handler)
```

where:

- *scope* identifies the object that Siebel Open UI uses to call the asynchronous method. You typically use the following scope:

```
this
```
- *done_handler* identifies the method that Siebel Open UI calls at the end of the asynchronous method that Siebel Open UI calls. The *done_handler* that Siebel Open UI registers with the callback method expects a return object. You use the `setReturnValue` method to return this object.

For example:

```
PharmaCallSubmitsvc.prototype.Submit = function () {  
    bc.ExecuteQuery();  
    $.callback(this, function(retObj){  
        err = retObj.err;  
    });  
}
```

For another example that uses this method, [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#).

eachAsyncOp Method

The `eachAsyncOp` method iteratively calls an asynchronous method. It uses the following syntax:

```
$.eachAsyncOp(scope, configObj)
```

where:

- *configObj* identifies the configuration object. A *configuration object* is a type of object that includes information that Siebel Open UI uses to send as input properties to a method.

For example:

```
$.eachAsyncOp(this, configObj);
```

The `eachAsyncOp` method handles the done handlers for each iteration. It requires a configuration object that includes the following properties as inputs:

- **executeScope**. Scope of the asynchronous method that Siebel Open UI must call.
- **execute**. Asynchronous method that Siebel Open UI must call.

- **preExecute.** Optional property that specifies the method that Siebel Open UI runs before it calls the asynchronous method. You can also use preExecute to send information that the asynchronous method requires. You must write this method so that it returns the following information:
 - Arguments that Siebel Open UI must send to the asynchronous method
 - Returns these arguments in an array.The preExecute property can use the iteration value as an input.
- **postExecute.** Optional property that specifies the method that Siebel Open UI runs after the asynchronous call finishes.
- **iterations.** Optional property that specifies the total number of iterations that eachAsyncOp runs. If you do not include this property, then Siebel Open UI runs the asynchronous method only one time.

SetErrorMsg Method

The SetErrorMsg method defines an error message for a business service that you customize. It returns nothing. It uses the following Syntax:

```
Siebel App. S_App. OffLineErrorObject.SetErrorMsg("messageKey", errParamArray);
```

where:

- *messageKey* contains the error message key. A *message key* is a text string that includes variable characters. %1 is an example of a variable character.
- *errParamArray* is an optional array that contains error properties that SetErrorMsg includes in the error message. SetErrorMsg replaces each variable character that the messageKey contains with a value from errParamArray.

For an example that uses SetErrorMsg, see [“Configuring Error Messages for Disconnected Clients” on page 369](#). For an example that uses SetErrorMsg in the context of a call to a custom business service, see [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#).

Other Methods You Can Use with Siebel Mobile Disconnected

This topic describes other methods that you can use with Siebel Mobile Disconnected. It includes the following topics:

- [“GetBusObj Method” on page 421](#)
- [“GetLovNameVal Method” on page 421](#)
- [“GetLovValName Method” on page 421](#)

GetBusObj Method

The GetBusObj method creates a new instance of a business object. It returns this new business object instance. It uses the following syntax:

```
Si ebel App. S_App. Model . GetBusObj (business_object_name)
```

where:

- *business_object_name* identifies the name of the business object that GetBusObj uses to create the new business object instance.

For example, the following code creates a new instance of the Service Request business object:

```
var pServiceRequestBC = Si ebel App. S_App. Model . GetBusObj ("Service Request")
```

The GetBusObj method resides in the model.js file.

You cannot configure Siebel Open UI to override this method.

GetLovNameVal Method

The GetLovNameVal method gets the value that Siebel Open UI currently displays in the client for a list of values. It uses the following syntax:

```
Si ebel App. S_App. Model . GetLovNameVal (LOV_name, LOV_type)
```

where:

- *LOV_name* identifies the name of a list of values.
- *LOV_type* identifies the type of list of values that *LOV_name* identifies.

For example, the following code gets the value that Siebel Open UI currently displays in the client for the Samples Request list of values:

```
Si ebel App. S_App. Model . GetLovNameVal ("Samples Request", "TODO_TYPE")
```

The GetLovNameVal method resides in the model.js file.

You cannot configure Siebel Open UI to override this method.

GetLovValName Method

The GetLovValName method gets the name of a value that resides in a list of values. It uses the following syntax:

```
Si ebel App. S_App. Model . GetLovVal Name(value_name, LOV_type)
```

where:

- *value_name* identifies the name of a value that resides in a list of values.
- *LOV_type* identifies the type of list of values that contains the value that *value_name* contains.

For example, the following code gets the value that Siebel Open UI currently displays in the client for the Call value:

```
Siebel App. S_App. Model . GetLovVal Name("Call ", "TODO_TYPE")
```

The GetLovValName method resides in the model.js file. You cannot configure Siebel Open UI to override this method.

A

Siebel Open UI Application Programming Interface

This appendix describes reference information for the JavaScript Application Programming Interface (API) that you can use to customize Siebel Open UI. It includes the following topics:

- [Overview of the Siebel Open UI Client Application Programming Interface on page 423](#)
- [Methods of the Siebel Open UI Application Programming Interface on page 423](#)
- [Methods for Pop-Up Objects, Google Maps, and Property Sets on page 508](#)

Overview of the Siebel Open UI Client Application Programming Interface

Creating a custom client user interface in Siebel Open UI requires that you do the following work:

- Creating a new presentation model that Siebel Open UI uses in addition to the metadata and data that it gets from the Web Engine that resides on the Siebel Server.
- Creating a new physical user interface by creating a custom physical renderer that Siebel Open UI uses in addition to a predefined or custom presentation model.

You can use the following programming interfaces to implement these presentation models:

- **Presentation model class.** Describes the life cycle methods that you must code for a presentation model and the control methods that Siebel Open UI uses to add presentation model properties and behavior. For more information, see [“Presentation Model Class” on page 424](#).
- **Physical renderer methods.** Describes the life cycle methods that you must code into any renderer that binds a presentation model to a physical renderer. For more information, see [“Physical Renderer Class” on page 460](#).

For a summary of these methods and information about how Siebel Open UI uses them, see [“Life Cycle of User Interface Elements” on page 54](#).

Siebel Open UI defines each class in a separate file. It stores these files in the following folder:

```
\release_number\EAPPWEB\PUBLIC\language_code\release_number\SCRIPTS\SIEBEL
```

For brevity, this chapter states that the method does *something*. In reality, most methods send a request to a proxy object, and then this proxy object does the actual work.

Methods of the Siebel Open UI Application Programming Interface

This topic describes the methods of the Siebel Open UI Application Programming Interface. You can use them to customize Siebel Open UI. It includes the following information:

- [Presentation Model Class on page 424](#)
- [Presentation Model Class for Applets on page 433](#)
- [Presentation Model Class for List Applets on page 452](#)
- [Presentation Model Class for Menus on page 458](#)
- [Physical Renderer Class on page 460](#)
- [Business Component Class on page 465](#)
- [Applet Class on page 465](#)
- [Applet Control Class on page 466](#)
- [JQ Grid Renderer Class for Applets on page 476](#)
- [Business Service Class on page 477](#)
- [Application Model Class on page 478](#)
- [Control Builder Class on page 489](#)
- [Locale Object Class on page 489](#)
- [Component Class on page 497](#)
- [Component Manager Class on page 500](#)
- [Cascading Style Sheet Classes on page 503](#)
- [Other Classes on page 506](#)

Presentation Model Class

This describes the methods that Siebel Open UI uses with the PresentationModel class. It includes the following information:

- [AddComponentCommunication Method on page 425](#)
- [AddMethod Method on page 425](#)
- [AddProperty Method on page 427](#)
- [AttachEventHandler Method on page 427](#)
- [AttachNotificationHandler Method on page 428](#)
- [AttachPMBinding Method on page 428](#)
- [AttachPostProxyExecuteBinding Method on page 429](#)
- [AttachPreProxyExecuteBinding Method on page 430](#)
- [ExecuteMethod Method on page 430](#)
- [Get Method on page 431](#)
- [Init Method on page 431](#)
- [OnControlEvents Method on page 432](#)

- [SetProperty Method on page 432](#)
- [Setup Method for Presentation Models on page 433](#)

Siebel Open UI defines the `PresentationModel` class in the `pmodel.js` file.

AddComponentCommunication Method

The `AddComponentCommunication` method binds a communication method. It uses the following arguments:

- *methodName* is a string that identifies the communication method that Siebel Open UI binds.
- *targetMethod* is a string that identifies the method that Siebel Open UI calls after *methodName* finishes. It calls this target method in the presentation model context.
- *targetMethodConfig* identifies an object that contains configuration properties for *targetMethod*.
- *targetMethodConfig.scope* identifies the object that the `AddComponentCommunication` method binds. This object must reference the *targetMethod*.
- *targetMethodConfig.args* is a list of arguments that Siebel Open UI sends to *targetMethod* when the `AddComponentCommunication` method runs.

AddMethod Method

The `AddMethod` method adds a method to a presentation model. You can use `ExecuteMethod` to run the method that `AddMethod` adds from the presentation model or from the physical renderer. If `AddMethod` attempts to add a new method that the predefined client already contains, then the new method becomes a customization of the predefined method, and this customization runs before or after the predefined method depending on the `CancelOperation` part of the return value.

A method that customizes another method can return to the caller without running the method that it customizes. To do this, you configure Siebel Open UI to set the `CancelOperation` part of the return value to true. You set this property on the `ReturnStructure` object that Siebel Open UI sends to each method as an argument. For an example that does this configuration, see [“Customizing the Presentation Model to Identify the Records to Delete” on page 66](#).

The `AddMethod` method returns one of the following values:

- **True.** Added a method successfully.
- **False.** Did not add a method successfully.

It uses the following syntax:

```
AddMethod("methodName", methodDef(argument, argument_n){
    }, {methodConfig : value});
```

where:

- *methodName* is a string that contains the name of the method that Siebel Open UI adds to the presentation model.
- *methodDef* is an argument that allows you to call a method or a method reference.

- *argument* and *argument_n* are arguments that `AddMethod` sends to the method that *methodDef* identifies.
- *methodConfig* is an argument that you set to one of the following values:
 - **sequence.** Set to one of the following values:
 - **true.** Siebel Open UI calls *methodName* before it calls the method that already exists in the presentation model.
 - **false.** Siebel Open UI calls *methodName* after it calls the method that already exists in the presentation model. The default value is false.
 - **override.** Set to one of the following values:
 - **true.** Siebel Open UI does not call the method that already exists in the presentation model. Instead, it calls the sent method, when necessary. Note that Siebel Open UI can never override some methods that exist in a predefined presentation model even if you set `override` to true.
 - **false.** Siebel Open UI calls the method that already exists in the presentation model.
 - **scope.** Describes the scope that Siebel Open UI must use when it calls *methodDef*. The default scope is Presentation Model.

Example of Adding a New Method

The following code adds a new `ShowSelection` method:

```
this.AddMethod("ShowSelection", SelectionChange, {sequence : false, scope : this});
```

After Siebel Open UI adds the `ShowSelection` method, you can use the following code to configure Siebel Open UI to call this method. It sends a string value of `SetActiveControl` to the sequence and a string value of `null` to the scope argument. To view how Siebel Open UI uses this example, see [Step 5 on page 68](#):

```
this.ExecuteMethod("SetActiveControl", null)
```

Example of Using the Sequence Argument

The following code configures Siebel Open UI to attach a method. It calls this method anytime it calls the `InvokeMethod` method of the proxy:

```
this.AddMethod("InvokeMethod", function(){  
    }, {sequence : true});
```

This code sets the sequence argument to true, which configures Siebel Open UI to call the method that it sends before it calls `InvokeMethod`. The method that it sends gets all the arguments that `InvokeMethod` receives. For more information, see [“InvokeMethod Method for Presentation Models” on page 443](#).

Example of Overriding the Predefined Presentation Model

The following example overrides the predefined presentation model and runs the `ProcessDrillDown` method:

```
this.AddMethod("ProcessDrillDown", function(){
}, {override : true});
```

Other Examples

The following examples also use AddMethod:

```
this.AddMethod("InvokeMethod", function(){console.log("In Invoke Method of PM"),
{override: true});
```

```
this.AddMethod("InvokeControlMethod",
DerivedPresentationModel.prototype.MyInvokeControlMethod, {sequence : true});
```

AddProperty Method

The AddProperty method adds a property to a presentation model. Siebel Open UI can access it through the Get method. It returns one of the following values:

- **True.** Added a property successfully.
- **False.** Did not add a property successfully.

It uses the following syntax:

```
this.AddProperty("propertyName", propertyValue);
```

where:

- *propertyName* is a string that identifies a property. A subsequent call to this method with the same *propertyName* overwrites the previous value.
- *propertyValue* assigns a value to the property.

For example, the following code adds the NumOfRows property and assigns a value of 10 to this property:

```
this.AddProperty("NumOfRows", 10);
SiebelJS.Log(this.Get("NumOfRows"));
```

AttachEventHandler Method

The AttachEventHandler method attaches an event handler for a physical event. It accepts the eventName and the corresponding event handler that Siebel Open UI attaches to this physical event. It returns one of the following values:

- **true.** Attached an event handler successfully.
- **false.** Did not attach an event handler successfully.

It uses the following syntax:

```
AttachEventHandler("eventName", eventHandler());
```

where:

- *eventName* is a string that identifies the name of the event that Siebel Open UI must attach to the physical event.

- *eventHandler* identifies the method that Siebel Open UI calls.

For an example that uses `AttachEventHandler`, see [“Example of the Life Cycle of a User Interface Element” on page 58](#).

You can use the `AttachEventHandler` method to override some of the predefined event handling. If you do this, then you must make sure it returns a value of `false`. For more information, see [“Overriding Event Handlers” on page 119](#).

For more information about events, see [“Siebel CRM Events You Can Use to Customize Siebel Open UI” on page 563](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

AttachNotificationHandler Method

The `AttachNotificationHandler` attaches a method that handles the notification that Siebel Open UI calls when the Siebel Server sends a notification to an applet. It does this attachment when the notification occurs. It returns one of the following values:

- **True.** Attached notification handler successfully.
- **False.** Did not attach notification handler successfully.

It uses the following syntax:

```
this.AttachNotificationHandler("notification_name", handler);
```

where:

- *notification_name* is a string that includes the name or type of a notification. For example, `NotifyDeleteRecord` or `SWE_PROP_BC_NOTI_DELETE_RECORD`.
- *handler* identifies a notification handler that Siebel Open UI calls when notification processing finishes. For example, `HandleDeleteNotification`.

For more information about:

- An example that uses `AttachNotificationHandler`, see [“Customizing the Presentation Model to Handle Notifications” on page 75](#)
- Notifications, see [“Notifications That Siebel Open UI Supports” on page 541](#)
- Using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#)

AttachPMBinding Method

The `AttachPMBinding` method binds a method to an existing method. Siebel Open UI calls the method that it binds when it finishes processing this existing method. The `AttachPMBinding` method returns one of the following values:

- **True.** The bind succeeded.
- **False.** The bind failed.

It uses the following syntax:

```
this.AttachPMBinding("method_name", function() { SiebelJS.Log("method_to_call"); }, { when : function(conditional_function) { return value; } });
```

where:

- *method_name* is a string that identifies the name of a method.
- *method_to_call* identifies the method that Siebel Open UI calls when it finishes processing *method_name*.
- *conditional_function* identifies a function that returns one of the following values:
 - **true**. Calls the AttachPMBinding method.
 - **false**. Does not call the AttachPMBinding method.

For an example that uses AttachPMBinding, see [“Customizing the Physical Renderer to Refresh the Recycle Bin” on page 89](#).

For more information about using the AttachPMBinding method, see [“Configuring Siebel Open UI to Bind Methods” on page 110](#) and [“Life Cycle Flows of User Interface Elements” on page 523](#).

AttachPostProxyExecuteBinding Method

The AttachPostProxyExecuteBinding method binds a method that resides in a proxy or presentation model to a PostExecute method. Siebel Open UI finishes the PostExecute method, and then calls the method that AttachPostProxyExecuteBinding identifies. It uses the following syntax:

```
this.AttachPostProxyExecuteBinding("method_to_call", function(methodName, inputPS, outputPS) { "binder_configuration"; return; });
```

where:

- *method_to_call* is a string that identifies the method that Siebel Open UI calls.
- *binder_configuration* is a string that identifies code that Siebel Open UI runs after the applet proxy sends a reply.

For more information, see [“Refreshing Custom Events” on page 118](#) and [“PostExecute Method” on page 445](#).

In the following example, the user clicks the New button in an applet, Siebel Open UI runs the NewRecord method, and then the client receives the reply from the Siebel Server. In this situation, you can use the following code to run some logic in the presentation model after Siebel Open UI runs the PostExecute method as part of the element life cycle:

```
this.AttachPostProxyExecuteBinding("NewRecord", function(methodName, inputPS, outputPS) { "Do Something for New Record"; return; });
```

The following code runs this same logic in the presentation model for all methods:

```
this.AttachPostProxyExecuteBinding("ALL", function(methodName, inputPS, outputPS) { "Do Something for all methods"; return; });
```

For more information, see [“NewRecord Method” on page 474](#).

For more examples that use `AttachPreProxyExecuteBinding` and `AttachPostProxyExecuteBinding`, see [“Customizing the Presentation Model to Call the Siebel Server and Delete a Record” on page 83](#) and [“Calling Methods for Applets” on page 111](#).

Using the `AttachPreProxyExecuteBinding` and `AttachPostProxyExecuteBinding` Methods

The `AttachPreProxyExecuteBinding` and `AttachPostProxyExecuteBinding` methods provide a generic way to do more processing than the `AttachNotificationHandler` method provides before or after the proxy finishes processing the reply from a method that the client or the Siebel Server calls. A method might cause Siebel Open UI to create a notification from the Siebel Server that does more post-processing than the client proxy requires. This situation can occur with a custom method that you implement on the Siebel Server. For example, with an applet, business service, or some other object type. For more information, see [“AttachNotificationHandler Method” on page 428](#).

Siebel Open UI sends a notification only for a typical modification that occurs in the predefined product. For example, a new or deleted record or a modified record set. Siebel Open UI might not be able to identify and process the correct notification. For example, you can configure Siebel Open UI to make one call to the `WriteRecord` method from the client, but the server business logic might cause this method to run more than one time. Siebel Open UI might receive notifications for any `WriteRecord` method that occurs for a business component that it binds to the current user interface. These notifications might contain more information than the reply notification requires. For more information, see [“WriteRecord Method” on page 407](#).

AttachPreProxyExecuteBinding Method

The `AttachPreProxyExecuteBinding` method binds a method that resides in a proxy or presentation model to a `PostExecute` method. Siebel Open UI calls this method, and then runs `PostExecute`. The `AttachPreProxyExecuteBinding` uses the same syntax and arguments that the `AttachPostProxyExecuteBinding` method uses, except you configure Siebel Open UI to call the `AttachPreProxyExecuteBinding` method. For more information, see [“AttachPostProxyExecuteBinding Method” on page 429](#).

ExecuteMethod Method

The `ExecuteMethod` method runs a method. You can use it to run a predefined or custom method that the presentation model contains. It makes sure Siebel Open UI runs all dependency chains for the method that it calls. For more information about dependency chains, see [“About Dependency Injection” on page 69](#).

If the method that `ExecuteMethod` specifies:

- **Exists.** It returns a value from the method that it specifies.
- **Does not exist.** It returns the following string:

undefi ned

It uses the following syntax:

```
this.s.GetPM().ExecuteMethod("method_name", arguments);
```

where:

- *method_name* is a string that identifies the name of the method that ExecuteMethod runs. You must use the AddMethod method to add the method that *method_name* specifies before you run ExecuteMethod. If the method that *method_name* specifies:
 - **Exists.** Siebel Open UI calls the method that *method_name* specifies, sends the arguments, gets the return value, and then sends this return value to the object that called the ExecuteMethod method.
 - **Does not exist.** The ExecuteMethod method does nothing.
- *arguments* includes a list of one or more arguments where a comma separates each argument. ExecuteMethod sends these arguments to the method that *method_name* specifies. It sends these arguments in the same order that you list them in this argument list.

For examples that use InvokeMethod, see [“Customizing the Presentation Model to Delete Records” on page 69](#) and [“Customizing the Presentation Model to Handle Notifications” on page 75](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

Get Method

The Get method returns the value of the property that Siebel Open UI adds through the AddProperty method. If Siebel Open UI sends a method in the propertyValue argument of the AddProperty method, then it calls the Get method, and then sends the return value to the method that calls the Get method. For an example that uses the Get method, see [“Customizing the Presentation Model to Delete Records” on page 69](#). For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

Init Method

The Init method allows you to use different methods to customize a presentation model, such as AddMethod, AddNotificationHandler, AttachPMBinding, and so on. It uses the following syntax:

```
Init()
```

For an example that uses Init, see [Step 2 on page 74](#).

You must not configure Siebel Open UI to override any method that resides in a derived presentation model except for the Init method or the Setup method.

You must configure Siebel Open UI to do the following:

- Call the Init method in the predefined presentation model before it calls the Init method in the derived presentation model.
- Call the Setup method in the predefined presentation model before it calls the Setup method in the derived presentation model. For more information, see [“Setup Method for Presentation Models” on page 433](#).

For more information about deriving values, see [“About Using This Book” on page 30](#).

OnControlEvent Method

The OnControlEvent method calls an event. It uses the following syntax:

```
OnControlEvent(event_name, event_arguments)
```

where:

- *event_name* identifies the name of an event. You must use *event_name* to send a physical event.

For more information about:

- Examples that use OnControlEvent, see the following topics:
 - [“Customizing the Event Handlers” on page 92](#)
 - [“Adding Custom User Preferences to Applets” on page 196.](#)
- How Siebel Open UI uses OnControlEvent, see the following topics:
 - [“How Siebel Open UI Uses the Init Method of the Presentation Model” on page 55](#)
 - [“Siebel CRM Events You Can Use to Customize Siebel Open UI” on page 563](#)
 - [“Life Cycle Flows of User Interface Elements” on page 523](#)

SetProperty Method

The SetProperty method sets the value of a presentation model property. It returns one of the following values:

- **True.** Set the property value successfully.
- **False.** Did not set the property value successfully.

It uses the following syntax:

```
SetProperty(property_name, property_value)
```

where:

- *property_name* specifies the name of the property that SetProperty sets.
- *property_value* specifies the value that SetProperty sets for *property_name*.

If the property that the SetProperty method references does not exist, then Siebel Open UI creates this property and sets the value for it according to the SetProperty method. You can also use the AddProperty method to add a property.

For examples that use SetProperty, see the following topics:

- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Customizing the Presentation Model to Call the Siebel Server and Delete a Record” on page 83](#)
- [“Text Copy of Code That Does a Partial Refresh for the Presentation Model” on page 161](#)
- [“Adding Custom User Preferences to Applets” on page 196](#)
- [“Using Custom JavaScript Methods” on page 364](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)

Setup Method for Presentation Models

The Setup method extracts the values that a property set contains. Siebel Open UI calls this Setup method when it processes the initial reply from the Siebel Server. It uses the following syntax:

```
Setup(property_set)
```

where:

- *property_set* identifies the property set that Siebel Open UI uses with the corresponding proxy object. It contains the property set information for the proxy and any custom property set information that Siebel Open UI added through the presentation model that resides on the Siebel Server. If Siebel Open UI must parse a custom property set, then this work must occur in the Setup method for the derived presentation model.

For example, the following code identifies the childPropset property set:

```
extObject.Setup(childPropset.GetChild(0));
```

For more information about:

- How Siebel Open UI uses this Setup method, see [“Summary of Presentation Model Methods” on page 54](#). [“GetChild Method” on page 520](#).
- Examples that use the Setup method, see [“Customizing the Setup Logic of the Presentation Model” on page 64](#) and [“Adding Presentation Model Properties That Siebel Servers Send for Applets” on page 106](#).
- Deriving values, see [“About Using This Book” on page 30](#).
- The Setup method that Siebel Open UI uses with components, see [“Setup Method for Components” on page 499](#).

Presentation Model Class for Applets

This topic describes the methods that Siebel Open UI uses with the presentation models that it uses to display applets. It includes the following information:

- [Summary of Methods That You Can Use with the Presentation Model for Applets on page 435](#)
- [Properties of the Presentation Model That Siebel Open UI Uses for Applets on page 436](#)
- [CanInvokeMethod Method for Presentation Models on page 437](#)
- [CanNavigate Method on page 438](#)
- [CanUpdate Method on page 439](#)
- [ExecuteMethod Method on page 439](#)
- [ExecuteUIUpdate Method on page 440](#)
- [FieldChange Method for Presentation Models on page 440](#)
- [FocusFirstControl Method on page 441](#)
- [GetControl Method on page 441](#)

- [GetControlId Method on page 442](#)
- [GetFieldValue Method on page 442](#)
- [GetFormattedFieldValue Method on page 442](#)
- [GetPhysicalControlValue Method on page 443](#)
- [InvokeMethod Method for Presentation Models on page 443](#)
- [InvokeStateChange Method on page 444](#)
- [IsPrivateField Method on page 444](#)
- [LeaveField Method on page 444](#)
- [NewFileAttachment Method on page 445](#)
- [PostExecute Method on page 445](#)
- [ProcessCancelQueryPopup Method on page 446](#)
- [RefreshData Method on page 446](#)
- [ResetAppletState Method on page 447](#)
- [SetActiveControl Method on page 447](#)
- [SetFocusDefaultControl Method on page 448](#)
- [SetHighlightState Method on page 448](#)
- [SetUpdateConditionals Method on page 448](#)
- [ShowPickPopup Method on page 449](#)
- [ShowPopup Method on page 449](#)
- [ShowSelection Method on page 449](#)
- [UpdateAppletMessage Method on page 450](#)
- [UpdateConditionals Method on page 450](#)
- [UpdateCurrencyCalcInfo Method on page 451](#)
- [UpdateQuickPickInfo Method on page 451](#)
- [UpdateStateChange Method on page 452](#)

Siebel Open UI uses the `PresentationModel` class to define the presentation models that it uses to display applets. For more information about this class, see [“Presentation Model Class” on page 424](#).

Summary of Methods That You Can Use with the Presentation Model for Applets

Table 26 lists the methods that you can use with the presentation model that Siebel Open UI uses for a predefined applet. You cannot configure Siebel Open UI to customize or override any of these methods except for the PostExecute method. You can configure Siebel Open UI to customize the PostExecute method.

Table 26. Summary of Methods That You Can Use with the Presentation Model for Applets

| Method | Callable | Bindable |
|-------------------------|----------|----------|
| CanInvokeMethod | Yes | No |
| CanNavigate | Yes | No |
| CanUpdate | Yes | No |
| ExecuteMethod | Yes | No |
| ExecuteUIUpdate | No | Yes |
| FieldChange | No | Yes |
| FocusFirstControl | No | Yes |
| GetControl | Yes | No |
| GetControlId | Yes | No |
| GetFieldValue | Yes | No |
| GetFormattedFieldValue | Yes | No |
| GetPhysicalControlValue | No | Yes |
| InvokeMethod | Yes | No |
| InvokeStateChange | No | Yes |
| IsPrivateField | Yes | No |
| LeaveField | Yes | No |
| NewFileAttachment | No | Yes |
| PostExecute | No | Yes |
| ProcessCancelQueryPopup | No | Yes |
| RefreshData | No | Yes |
| ResetAppletState | No | Yes |
| SetActiveControl | Yes | Yes |
| SetFocusDefaultControl | Yes | No |
| SetFocusToCtrl | No | Yes |
| SetHighlightState | No | Yes |

Table 26. Summary of Methods That You Can Use with the Presentation Model for Applets

| Method | Callable | Bindable |
|------------------------|----------|----------|
| SetUpdateConditionals | Yes | No |
| ShowPickPopup | Yes | No |
| ShowPopup | No | Yes |
| ShowSelection | No | Yes |
| UpdateAppletMessage | No | Yes |
| UpdateConditionals | No | Yes |
| UpdateCurrencyCalcInfo | No | Yes |
| UpdateQuickPickInfo | No | Yes |
| UpdateStateChange | No | Yes |

Properties of the Presentation Model That Siebel Open UI Uses for Applets

Table 27 lists the properties of the presentation model that Siebel Open UI uses for applets.

Table 27. Properties of the Presentation Model That Siebel Open UI Uses for Applets

| Property | Description |
|------------------------|--|
| GetActiveControl | Returns a string that identifies the active control of the applet for the presentation model. |
| GetAppleLabel | Returns a string that includes the applet label. |
| GetAppletSummary | Returns a string that includes the applet summary. |
| GetControls | Returns an array that lists control objects that the applet includes for the presentation model. |
| GetDefaultFocusOnNew | Returns a string that identifies the control name where Siebel Open UI must set the default focus when the user creates a new record in an applet. |
| GetDefaultFocusOnQuery | Returns a string that identifies the control name where Siebel Open UI must set the default focus when the user runs a query in the applet. |
| GetFullId | Returns a string that includes the Applet Full Id that the Siebel Server sends for the presentation model. |
| GetId | Returns a string that includes the applet ID that the Siebel Server sends for the presentation model. For an example usage of this property, see “Customizing the Physical Renderer to Render List Applets” on page 86 . |
| GetMode | Returns a string that identifies the applet mode. |

Table 27. Properties of the Presentation Model That Siebel Open UI Uses for Applets

| Property | Description |
|--------------------|---|
| GetName | Returns a string that includes the name of the presentation model. |
| GetPrsrvControl | Returns a string that identifies the control object of a preserve control that Siebel Open UI sets in a leave field. |
| GetQueryModePrompt | Returns a string that identifies the prompt that the applet uses when it is in query mode. |
| GetRecordset | Returns an array that lists the record set that the applet currently contains. |
| GetSelection | Returns the number of records the user chooses in the applet. |
| GetTitle | Returns a string that includes the applet title of the presentation model. |
| GetUIEventMap | <p>Returns an array that lists pending user interface events. Each element in this array identifies an event that you can access using the following code:</p> <pre>thi s. Get("GetUI EventMap") [i ndex]. ev</pre> <p>You can use the following code to access the arguments:</p> <pre>as th i s. Get("GetUI EventMap") [i ndex]. ar</pre> |
| IsInQueryMode | Returns a Boolean value that indicates if the applet is in query mode. |
| IsPure | Returns a Boolean value that indicates if the applet has Buscomp. |

Adding Code to the Physical Renderer

You add code for some methods to the section of code in the physical renderer that binds the control to the presentation model. For example, if you must customize code for a currency calculator control, then you modify the code in the physical renderer that binds the currency calculator control to the presentation model. This appendix indicates the methods that must use this configuration.

CanInvokeMethod Method for Presentation Models

The CanInvokeMethod method that Siebel Open UI uses for presentation models determines whether or not Siebel Open UI can invoke a method. It returns one of the following values:

- **true.** Siebel Open UI can invoke the method.
- **false.** Siebel Open UI cannot invoke the method.

It uses the following syntax:

```
CanI nvokeMethod(method_name)
```

where:

- *method_name* is a string that contains the name of the method that `CanInvokeMethod` examines. You must enclose this string in double quotation marks, or use a literal value of `methodName`.

For example, you can add the following code in a physical renderer to determine whether or not Siebel Open UI can call the method that *method_name* specifies, and if it can call this method on the control that *control* specifies:

```
var controlSet = this.GetPM().Get("GetControls");
for(var control in controlSet){
    if(controlSet.hasOwnProperty(control)){
        var canInvoke = this.GetPM().ExecuteMethod("CanInvokeMethod", controlSet[
            control ].GetMethodName());
    }
}
```

To avoid an error on the Siebel Server, it is recommended that you configure Siebel Open UI to use `CanInvokeMethod` immediately before it uses `InvokeMethod` to make sure it can call the method.

For information about the `CanInvokeMethod` method that Siebel Open UI uses for application models, see [“CanInvokeMethod Method for Application Models” on page 479](#).

For more examples that use `CanInvokeMethod`, see the following topics:

- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Attaching an Event Handler to a Presentation Model” on page 78](#)
- [“Customizing Applets to Capture Signatures” on page 199](#)
- [“Customizing a Resource Scheduler” on page 217](#)
- [“Using Custom JavaScript Methods” on page 364](#)
- [“Using Custom Siebel Business Services” on page 367](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Commit Part Tracker Records” on page 376](#)

CanNavigate Method

The `CanNavigate` method determines whether or not the user can navigate a control. It returns one of the following values:

- **true.** The user can navigate the control.
- **false.** The user cannot navigate the control.

It uses the following syntax:

```
CanNavigate(activeControl.GetFieldName())
```

For example, the following code uses the `CanNavigate` method to set up a variable named `canNavigate`:

```
var controlSet = this.GetPM().Get("GetControls");
for(var control in controlSet){
    if(controlSet.hasOwnProperty(control)){
```

```

        var canNavigate = this.s.GetPM().ExecuteMethod("CanNavigate", controlSet[
        control ].GetName());
    }
}

```

The following example identifies the controls in a set of controls that reside in an applet proxy. You can then use the value that CanNavigate returns to determine whether or not Siebel Open UI can render a control as a link:

```

var controlSet = this.s.GetPM().Get("GetControls");
for(var control in controlSet){
    if(controlSet.hasOwnProperty(control)){
        var canNavigate = this.s.GetPM().ExecuteMethod("CanNavigate", controlSet[
        control ].GetName());
    }
}

```

CanUpdate Method

The CanUpdate method determines whether or not Siebel Open UI can update a control. It returns one of the following values:

- **true.** The user can update the control.
- **false.** The user cannot update the control.

It uses the following syntax:

```
CanUpdate(control_name)
```

where:

- *control_name* identifies the name of the control that CanUpdate examines.

The following example identifies the controls that exist in a set of controls that reside in an applet proxy. You can then use the value that CanUpdate returns to write custom code in the physical renderer that modifies a control that Siebel Open UI can update:

```

var controlSet = this.s.GetPM().Get("GetControls");
for(var control in controlSet){
    if(controlSet.hasOwnProperty(control)){
        var canupdate = this.s.GetPM().ExecuteMethod("CanUpdate", controlSet[ control
        ].GetName());
    }
}

```

For an example that uses the CanUpdate method, see ["UpdateRecord Method" on page 407](#).

ExecuteMethod Method

The ExecuteMethod method runs a method that is available in the presentation model. It returns nothing. It uses the following syntax:

```
ExecuteMethod("method_name", arguments);
```

where:

- *method_name* is a string that identifies the name of the method that ExecuteMethod runs.
- *arguments* lists the arguments that Siebel Open UI requires to run the method that *method_name* identifies.

For examples that use ExecuteMethod, see the following topics:

- [“Customizing the Presentation Model to Identify the Records to Delete” on page 66.](#)
- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Customizing the Presentation Model to Handle Notifications” on page 75](#)
- [“Calling Methods for Applets” on page 111](#)
- [“Accessing Proxy Objects” on page 117](#)
- [“Allowing Users to Interact with Clients During Business Service Calls” on page 120](#)
- [“Customizing List Applets to Render as a Carousel without Compiling the SRF” on page 173](#)

For information about how Siebel Open UI uses the ExecuteMethod method, see [“How Siebel Open UI Uses the Init Method of the Presentation Model” on page 55.](#)

ExecuteUIUpdate Method

The ExecuteUIUpdate method updates objects in the user interface. It uses the following syntax:

```
ExecuteUIUpdate()
```

For example, the following code in the applicationcontext.js file updates objects that reside in the current applet:

```
applet.ExecuteUIUpdate();
```

You can configure Siebel Open UI to call the ExecuteUIUpdate method in the following ways:

- In the physical renderer:

```
this.GetPM().AttachPMBinding("ExecuteUIUpdate", function(){  
    custom_code  
});
```

- In the presentation model:

```
this.AddMethod("ExecuteUIUpdate", function(){  
    custom_code  
}, {sequence: true, scope: this});
```

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437.](#)

FieldChange Method for Presentation Models

The FieldChange method that Siebel Open UI uses with presentation models modifies the value of a field. It returns nothing. It uses the following syntax:


```
FieldChange(control, field_value)
```

where:

- *control* identifies the name of a control.
- *field_value* is a modified value of the control.

For example, you can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("FieldChange", function(control, field_value){  
    custom_code  
});
```

where:

- *custom_code* is code that you write that sets the value for the control.

For more information about:

- Where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#)
- An example that uses the FieldChange method, [“Refreshing Applets That Contain Modified Data” on page 159](#)
- Using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#)
- The FieldChange method that Siebel Open UI uses with physical renderers, see [“FieldChange Method for Physical Renderers” on page 463](#)

FocusFirstControl Method

The FocusFirstControl method sets the focus on the first control that the presentation model displays. It uses the following syntax:

```
FocusFirstControl()
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("FocusFirstControl", function(){  
    custom_code;  
});
```

where:

- *custom_code* is code that you write that handles focus updates from the Siebel Server. For example, updating the enable or disable state of a user interface control that the UpdateUIButtons method of the physical renderer specifies. For more information about the UpdateUIButtons method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

GetControl Method

The GetControl method returns a control instance. It uses the following syntax:

```
GetControl (control_name)
```

where:

■ *control_name* identifies the name of the control that GetControl gets.

You add this code to the physical renderer.

For examples that use GetControl, see the following topics:

■ [“Customizing Control User Properties for Presentation Models” on page 108](#)

■ [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)

GetControlId Method

The GetControlId method gets the control ID of a toggle applet. It uses the following syntax:

```
GetControlId()
```

For example, the following code gets the control ID of the toggle applet that Siebel Open UI currently displays in the client. This code resides in the applet.js file:

```
return this.GetToggleApplet().GetControlId();
```

You can add the following code to the physical renderer:

```
var ToggleEI = this.GetPM().ExecuteMethod("GetControlId");
```

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

GetFieldValue Method

The GetFieldValue method returns the value of a field. It uses the following syntax:

```
this.GetFieldVal ue(field_name);
```

where:

■ *field_name* identifies the name of a field.

For example, the following code gets the current value of the Call Status field:

```
pBusComp.GetFieldVal ue("Call Status");
```

For another example that uses the GetFieldValue method, see [“Text Copy of Code That Does a Partial Refresh for the Presentation Model” on page 161](#).

GetFormattedFieldValue Method

The GetFormattedFieldValue method gets the formatted field value of a control. It uses the following syntax:

```
val ue = this.GetPM().ExecuteMethod("GetFormattedFi el dVal ue", control_name, flag,  
index);
```

where:

- *control_name* identifies the name of the control.
- *flag* is one of the following values:
 - **true**. Get the formatted field value from the work set.
 - **false**. Do not get the formatted field value from the work set.
- *index* is an index of the record set.

For an example that uses the `GetFormattedFieldValue` method, see [“Overriding Predefined Methods in Presentation Models” on page 74](#).

You add the `GetFormattedFieldValue` method to the physical renderer.

Siebel Open UI gets the formatted field value according to the locale object. For example, 1000 is an unformatted value, and 1,000 is a formatted value.

GetPhysicalControlValue Method

The `GetPhysicalControlValue` method gets the value of a physical control. It uses the following syntax:

```
GetPhysicalControlValue (control);
```

For example, the following code gets the value of the physical control that `control` identifies. This code resides in the `pmodel.js` file:

```
this.GetRenderer().GetPhysicalControlValue(control);
```

The following example binds the physical renderer to the presentation model. You add this code to the physical renderer:

```
this.AttachPMBinding("GetPhysicalControlValue", function(control){  
    custom_code  
});
```

where:

- *control* identifies the control value that Siebel Open UI must get from the physical counterpart of this control from the presentation model.
- *custom_code* is code that you write that gets the value from the physical control.

InvokeMethod Method for Presentation Models

The `InvokeMethod` method that Siebel Open UI uses for presentation models calls a method on the applet proxy. It is similar to the `InvokeMethod` method that Siebel Open UI uses for application models. For more information, see [“InvokeMethod Method for Application Models” on page 485](#).

InvokeStateChange Method

The `InvokeStateChange` method invokes a state change. It allows you to configure Siebel Open UI to handle updates. Siebel Open UI calls it when it sends a can invoke notification update from the Siebel Server. The `InvokeStateChange` method uses the following syntax:

```
InvokeStateChange()
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("InvokeStateChange", function(){  
    custom_code;  
});
```

where:

- *custom_code* is code that you write that handles updates from the Siebel Server. For example, updating the focus state of a user interface control that the `UpdateUIButtons` method of the physical renderer specifies. For more information about the `UpdateUIButtons` method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

IsPrivateField Method

The `IsPrivateField` method determines whether or not a field is private. A *private field* is a type of field that only allows the record owner to view the record. For more information about private fields, see *Siebel Object Types Reference*.

The `IsPrivateField` method returns one of the following values:

- **true**. The field that the control references is private.
- **false**. The field that the control references is not private.

It uses the following syntax:

```
this.IsPrivateField(control.GetName())
```

You add the following code in the physical renderer:

```
var bPvtField = this.GetPM().ExecuteMethod("IsPrivateField", control.GetName());
```

LeaveField Method

The `LeaveField` method determines whether or not Siebel Open UI has removed the focus from a field in an applet. It returns one of the following values:

- **true**. Siebel Open UI has removed the focus from a field. This situation typically occurs when the user navigates away from the field. To do this navigation, the user clicks another object in the applet or navigates away from the applet.
- **false**. Siebel Open UI has not removed the focus from a field.

It uses the following syntax:

```
LeaveField(control, value, do_not_leave);
```

where:

- *control* identifies the control that LeaveField examines.
- *value* contains the value that Siebel Open UI sets in the proxy for the control.
- *do_not_leave* is set to one of the following values:
 - **true**. Keep the focus on the control.
 - **false**. Do not keep the focus on the control.

For examples that use the LeaveField method, see [“Customizing the Presentation Model to Identify the Records to Delete” on page 66](#) and [“Customizing Methods in the Presentation Model to Store Field Values” on page 81](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

NewFileAttachment Method

The NewFileAttachment method returns the properties of a file attachment. It uses the following syntax:

```
var attdata = this.GetPM().ExecuteMethod ("NewFileAttachment");
```

It includes no arguments.

PostExecute Method

The PostExecute method runs in the presentation model after the InvokeMethod method finishes running. Siebel Open UI calls the InvokeMethod method, returns the call from the Siebel Server, and then runs PostExecute. The PostExecute method uses the following syntax:

```
PostExecute(cmd, inputPS, menuPS, lpcsa);
```

You add this code in the presentation model:

```
this.AddMethod("PostExecute", function(method_name, input_property_set,
output_property_set)
{ custom_code },
{sequence : true, scope : this});
```

where:

- *method_name* identifies the method that the Siebel Server called from the applet proxy.
- *input_property_set* contains the property set that Siebel Open UI sends to the Siebel Server from the applet proxy.
- *output_property_set* contains the property set that Siebel Open UI sends from the Siebel Server to the applet proxy.

■ *custom_code* is code that you write that customizes a PostExecute method.

For an example that uses the PostExecute method, see [“Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence” on page 359](#).

For more information about using this method, see [“AttachPostProxyExecuteBinding Method” on page 429](#) and [“Life Cycle Flows of User Interface Elements” on page 523](#).

ProcessCancelQueryPopup Method

The ProcessCancelQueryPopup method cancels a query dialog box if the user clicks Cancel in this dialog box. It uses the following syntax:

```
ProcessCancelQueryPopup()
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("ProcessCancelQueryPopup", function(){custom_code},  
{scope : this});
```

where:

■ *custom_code* is code that you write that cancels the query dialog box.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

RefreshData Method

The RefreshData method is proxy method that Siebel Open UI calls when it refreshes an applet in the client according to data that the applet proxy contains. It returns nothing. It uses the following syntax:

```
RefreshData(value)
```

where:

■ *value* contains one of the following values:

- **true**. Refresh the applet.
- **false**. Do not refresh the applet.

For example, the following code refreshes the current applet. It resides in the applet.js file:

```
this.RefreshData(true);
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("RefreshData", function(value){  
    custom_code});
```

where:

■ *value* contains one of the following values:

- **true**. Refresh the applet.

- **false**. Do not refresh the applet.

- *custom_code* is code that you write that refreshes data in the client user interface.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

ResetAppletState Method

The ResetAppletState method sets the applet to an active state if this applet is not active. It uses the following syntax:

```
oldActiveApplet.ResetAppletState();
```

It includes no arguments.

To use the ResetAppletState method, you bind the physical renderer to the presentation model. The following example binds the physical renderer to the presentation model. You add this code to the physical renderer:

```
this.s.GetPM().AttachPMBinding("ResetAppletState", function(){  
    //Code that resets the applet  
})  
});
```

SetActiveControl Method

The SetActiveControl method sets the active property of a control in an applet. It returns nothing. It uses the following syntax:

```
this.s.ExecuteMethod("SetActiveControl", control_name);
```

where:

- *control_name* identifies the name of a control.

The following code in the presentation model sets the active control to null so that the applet contains no active control:

```
this.s.ExecuteMethod("SetActiveControl", null);
```

For examples that use the SetActiveControl method, see the following topics:

- [“Customizing the Presentation Model to Identify the Records to Delete” on page 66](#)
- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Accessing Proxy Objects” on page 117](#)
- [“AddMethod Method” on page 425](#)

The predefined Siebel Open UI code handles an active control for the applet, so it is recommended that you do not configure Siebel Open UI to directly call the SetActiveControl method. You can use SetActiveControl only in the context of another call that Siebel Open UI makes to an applet control.

SetHighlightState Method

The SetHighlightState method sets the highlight for the active applet. It uses the following syntax:

```
SetHighlightState(i sActive, newActiveApplet)
```

For example:

```
this.SetHighlightState(i sActive);
```

You can add the following code to the physical renderer:

```
this.AttachPMBinding("SetHighlightState", function(i sActive, newActiveApplet){  
    custom_code  
});
```

where:

■ *custom_code* is code that you write that sets the highlight.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

SetFocusDefaultControl Method

The SetFocusDefaultControl method sets the default focus flag.

SetUpdateConditionals Method

Siebel Open UI calls the SetUpdateConditionals method when the Siebel Server sends change selection information or Can Invoke method notifications to the client. It uses the following syntax:

```
this.SetUpdateConditionals(condition);
```

where:

■ *condition* is true or false.

For example, the following code resides in the applet.js file:

```
this.SetUpdateConditionals(true);
```

You can add the following code in the physical renderer to the end of the UpdateConditionals method. This placement makes sure Siebel Open UI runs UpdateConditionals before it runs SetUpdateConditionals:

```
this.GetPM().ExecuteMethod("SetUpdateConditionals", false);
```

For more information, see [“Notifications That Siebel Open UI Supports” on page 541](#).

ShowPickPopup Method

The ShowPickPopup method displays the currency pick applet when the user clicks a pick icon in a currency calculator control. It uses the following syntax:

```
ShowPickPopup();
```

For example, the applet.js file includes the following code:

```
return this.GetCurrencyApplet().ShowPickPopup(this);
```

You can use the following code:

```
this.GetPM().ExecuteMethod("ShowPickPopup");
```

ShowPopup Method

The ShowPopup method displays a dialog box for a calculator control, date control, or date-time control. It uses the following syntax:

```
ShowPopup()
```

For example, the applet.js file includes the following code:

```
this.ShowPopup(control);
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("ShowPopup", function(control){  
    predefined_code;  
}, {scope : this});
```

where:

- *predefined_code* is code that exists in the physical renderer that you reuse to display the dialog box

ShowSelection Method

The ShowSelection method makes a record the active record. It does not return any values. It uses the following syntax:

```
ShowSelection()
```

It includes no arguments.

For example, the pmodel.js file includes the following code:

```
this.GetApplet(strAppletName).ShowSelection();
```

It uses the following code to bind the presentation model in the physical renderer:

```
this.GetPM().AttachPMBinding("ShowSelection", function(){custom_code});
```

where:

- *custom_code* is code that you write. Siebel Open UI runs the ShowSelection method that the applet proxy calls, and then runs your custom code. You add this custom code to the physical renderer.

For examples that use the ShowSelection method, see [“Text Copy of Code That Does a Partial Refresh for the Presentation Model” on page 161](#) and [“Example of Adding a New Method” on page 426](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

UpdateAppletMessage Method

The UpdateAppletMessage method updates an applet message according to modifications that exist on the Siebel Server. It uses the following syntax:

```
UpdateAppletMessage()
```

For example, the applet.js file includes the following code:

```
this.UpdateAppletMessage();
```

You add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("UpdateAppletMessage", function(){custom_code},  
{scope: this});  
//e.g. UpdateAppletMessageUI in phyrenderer.
```

where:

- *custom_code* is code that you write that displays a message.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

UpdateConditionals Method

The UpdateConditionals method runs when Siebel Open UI displays an applet. It uses the following syntax:

```
UpdateConditionals()
```

For example, the listapplet.js file contains the following code:

```
this.UpdateConditionals();
```

You can add the following code to the code that updates the physical properties and the HTML properties of the control:

```
this.GetPM().AttachPMBinding("UpdateConditionals", function(){custom_code}, {scope:  
: this});
```

where:

- *custom_code* is code that you write that updates HTML controls. Siebel Open UI runs this code as soon as the proxy calls the UpdateConditionals method.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

UpdateCurrencyCalcInfo Method

The UpdateCurrencyCalcInfo method updates information that Siebel Open UI uses for a currency calculation. Siebel Open UI calls it when it sends currency information from the Siebel Server. You can use it to display currency information in an applet. It uses the following syntax:

```
UpdateCurrencyCalcInfo(0, args)
```

For example, the applet.js file contains the following code:

```
this.UpdateCurrencyCalcInfo(0, args);
```

You can add the following code to the InvokeCurrencyApplet method of the physical renderer:

```
this.GetPM().AttachPMBinding("UpdateCurrencyCalcInfo", function(){custom_code},  
    {scope: this});
```

where:

- *custom_code* is code that you write that updates information in the currency calculator control.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

UpdateQuickPickInfo Method

The UpdateQuickPickInfo method updates List of Values (LOV) information. Siebel Open UI calls it when it sends LOV information from the Siebel Server to the client. It uses the following syntax:

```
UpdateQuickPickInfo(field, true, arrValues, 0);
```

For example:

```
this.UpdateQuickPickInfo(field, true, arrValues, 0);
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("UpdateQuickPickInfo", function(){custom_code},  
    {scope: this});
```

where:

- *custom_code* is code that you write that updates information in the LOV.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

UpdateStateChange Method

The UpdateStateChange method handles notification updates. Siebel Open UI calls it when it sends notification updates from the Siebel Server. It uses the following syntax:

```
UpdateStateChange()
```

You can add the following code to the physical renderer:

```
this.GetPM().AttachPMBinding("UpdateStateChange", function(){  
    custom_code;  
});
```

where:

- *custom_code* is code that you write that handles state change updates from the Siebel Server. For example, updating the enable or disable state of a user interface control that the UpdateUIControls method of the physical renderer specifies.

For information about where you add this code, see [“Adding Code to the Physical Renderer” on page 437](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#) and [“Notifications That Siebel Open UI Supports” on page 541](#).

Presentation Model Class for List Applets

This topic describes the methods that Siebel Open UI uses with the presentation models that it uses to display list applets. It includes the following information:

- [Properties of the Presentation Model That Siebel Open UI Uses for List Applets on page 453](#)
- [Summary of Methods That You Can Use with the Presentation Model That Siebel Open UI Uses for List Applets on page 454](#)
- [CellChange Method on page 454](#)
- [HandleRowSelect Method on page 455](#)
- [OnClickSort Method on page 455](#)
- [OnCtrlBlur Method on page 456](#)
- [OnCtrlFocus Method on page 456](#)
- [OnDrillDown Method on page 457](#)
- [OnVerticalScroll Method on page 457](#)
- [SetMultiSelectMode Method on page 458](#)

The presentation model that Siebel Open UI uses for list applets uses the ListPresentationModel class, which is a subclass of the class that Siebel Open UI uses with the presentation models that display applets. Siebel Open UI defines this presentation model in the listpmodel.js file. For more information about the class that Siebel Open UI uses with the presentation models that display applets, see [Presentation Model Class for Applets on page 433](#).

Properties of the Presentation Model That Siebel Open UI Uses for List Applets

Table 28 lists the properties of the presentation model that Siebel Open UI uses for a list applet.

Table 28. Properties of the Presentation Model That Siebel Open UI Uses for List Applets

| Property | Description |
|----------------------|---|
| GetBeginRow | Returns the beginning row number of a list applet. |
| GetListOfColumns | <p>Returns an array, where each item in this array corresponds to a column control in a list applet. Each of these items is defined as a JSON object with the following keys:</p> <ul style="list-style-type: none"> ■ name ■ controlType ■ isLink ■ index ■ bCanUpdate ■ control <p>For more information about JSON, see the JSON Web site at http://www.json.org.</p> |
| GetRowIdentifier | Returns a string that contains information about the row. |
| GetRowListRowCount | Returns the number of rows that a list applet contains. |
| GetRowsSelectedArray | <p>Returns an array, where each item in this array corresponds to a row number in a list applet. Each array item includes one of the following values:</p> <ul style="list-style-type: none"> ■ true. The row is chosen. ■ false. The row is not chosen. |
| HasHierarchy | Returns a Boolean value that indicates whether or not the list can include hierarchical records. |

Summary of Methods That You Can Use with the Presentation Model That Siebel Open UI Uses for List Applets

Table 29 summarizes the methods that you can use with the presentation model that Siebel Open UI uses for a list applet. You cannot configure Siebel Open UI to customize or override any of these methods.

Table 29. Summary of Methods That You Can Use with the Presentation Model That Siebel Open UI Uses for List Applets

| Method | Callable | Bindable |
|--------------------|----------|----------|
| CellChange | No | Yes |
| HandleRowSelect | Yes | Yes |
| OnClickSort | Yes | No |
| OnCtrlBlur | Yes | No |
| OnCtrlFocus | Yes | No |
| OnDrillDown | Yes | No |
| OnVerticalScroll | Yes | No |
| SetMultiSelectMode | No | Yes |

CellChange Method

The CellChange method determines whether or not Siebel Open UI modified the value of a control. If Siebel Open UI modified this value, then it returns the new value. It uses the following syntax:

```
CellChange(rowId, field_name, value);
```

where:

- *rowId* is a number of zero through *n*, where *n* is the maximum number of rows that the list applet displays. This number identifies the row that contains the control.
- *field_name* identifies the name of the control that Siebel Open UI analyzes to determine whether or not Siebel Open UI modified the value.
- *value* is a value that the control contains.

For example, the following code from the listapplet.js file determines whether or not Siebel Open UI modified the value of a control. The GetName method identifies this control. The *value* argument is a variable that contains the control value:

```
this.CellChange(rowId, control.GetName(), value);
```

Siebel Open UI can bind the physical renderer to the CellChange method to determine whether or not it modified the value for the control.

HandleRowSelect Method

The HandleRowSelect method chooses a row. It returns one of the following values:

- **true**. Row chosen successfully.
- **false**. Row not chosen due to an error in the client or on the Siebel Server.

It uses the following syntax:

```
HandleRowSelect(rowId, control_key, shift_key);
```

where:

- *rowId* is a number of zero through *n*, where *n* is the maximum number of rows that the list applet displays. This number identifies the row that HandleRowSelect chooses.
- *control_key* is one of the following values:
 - **true**. Choose the CTRL key when choosing the row.
 - **false**. Do not choose the CTRL key when choosing the row.
- *shift_key* is one of the following values:
 - **true**. Choose the SHIFT key when choosing the row.
 - **false**. Do not choose the SHIFT key when choosing the row.

For an example that uses HandleRowSelect, see [“Customizing the Presentation Model to Delete Records” on page 69](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

OnClickSort Method

The OnClickSort method sorts a column. It uses the following syntax:

```
OnClickSort(name, direction);
```

where:

- *name* identifies the name of the control that Siebel Open UI sorts.
- *direction* is one of the following values:
 - **asc**. Sort the control in ascending order.
 - **desc**. Sort the control in descending order.

For example, the following code sorts the my_accounts control in descending order:

```
bReturn = this.GetProxy().OnClickSort(my_accounts, desc);
```

For another example, the following code sorts the my_accounts control in ascending order:

```
this.ExecuteMethod("OnClickSort", my_accounts, asc);
```

This method is asynchronous, so you cannot configure Siebel Open UI to bind it. For more information, see [“About Synchronous and Asynchronous Requests” on page 74](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

OnCtrlBlur Method

The OnCtrlBlur method blurs a control, where *blur* is a state that makes the control not the active control. It returns nothing. It uses the following syntax:

```
OnCtrlBlur(rowId, control, value);
```

where:

- *rowId* is a number of zero through *n*, where *n* is the maximum number of rows that the list applet displays. This number identifies the row that contains the control.
- *control* identifies the control that Siebel Open UI must blur.
- *value* is a variable that contains the value of the control.

For example, the following code blurs the my_accounts control. This control resides in the row that the counter variable identifies. For example, if the counter variable contains a value of 3, then OnCtrlBlur blurs the my_accounts control that resides in row 3. The *value* argument is a variable that contains the control value. For example, if the value of the my_accounts control is Oracle, then the *value* variable contains a value of Oracle:

```
this.ExecuteMethod("OnCtrlBlur", counter, my_accounts, value);
```

OnCtrlBlur does the localization and notifies the binder method that Siebel Open UI attaches through the CellChange method, when required. If Siebel Open UI configures the control to do ImmediatePostChanges, then OnCtrlBlur also runs these modifications.

You must make sure Siebel Open UI uses the OnCtrlFocus method to make the control active before you use the OnCtrlBlur method. If the control is not active, then Siebel Open UI rejects any OnCtrlBlur call. For more information, see [“OnCtrlFocus Method” on page 456](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

OnCtrlFocus Method

The OnCtrlFocus method brings a control into focus, where *focus* is a state that makes the control the active control. It uses the following syntax:

```
OnCtrlFocus(rowId, control, value);
```

where:

- *rowId*, *control*, and *value* work the same as with the OnCtrlBlur method.

For example, the following code brings the my_accounts control into focus:

```
this.ExecuteMethod("OnCtrlFocus", counter, my_accounts, value);
```

For more information about these arguments and this example, see [“OnCtrlBlur Method” on page 456](#).

You must make sure no other control is active. If another control is already active, and if you configure Siebel Open UI to run `OnCtrlFocus`, then Siebel Open UI rejects the `OnCtrlFocus` call.

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

OnDrillDown Method

The `OnDrillDown` method drills down on a control. It returns one of the following values:

- **true**. Drilldown succeeded.
- **false**. Drilldown failed because an error occurred on the client or on the Siebel Server.

It uses the following syntax:

```
OnDrillDown(control_name, rowId);
```

where:

- *control_name* identifies the name of the control where Siebel Open UI does the drilldown.
- *rowId* is a number of zero through *n*, where *n* is the maximum number of rows that the list applet displays. This number identifies the row that contains the control where Siebel Open UI does the drilldown.

For example, the following code drills down on the `my_accounts` control. The counter identifies the row that contains this control. For more information about how another example uses this counter, see [“OnCtrlBlur Method” on page 456](#):

```
this.ExecuteMethod("OnDrillDown", my_accounts, counter);
```

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

OnVerticalScroll Method

The `OnVerticalScroll` method scrolls a set of records. It returns nothing. It uses the following syntax:

```
OnVerticalScroll(scroll_action);
```

where:

- *scroll_action* is one of the following values:
 - **nxrc**. Scroll down to the next record.
 - **pvr**. Scroll up to the previous record.
 - **pgdn**. Page down to the next set of records.
 - **pgup**. Page up to the prior set of records.

For example, the following code configures Siebel Open UI to scroll to the next record. You add this code to the physical renderer:

```
this.ExecuteMethod("OnVerticalScroll", "nxrc");
```

This method is asynchronous, so you cannot configure Siebel Open UI to bind it. For more information, see [“About Synchronous and Asynchronous Requests” on page 74](#).

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

SetMultiSelectMode Method

The SetMultiSelectMode method determines whether or not a list applet is using multiselect mode. It uses the following syntax:

```
SetMultiSelectMode(bInMultiSelectMode)
```

where:

- **bInMultiSelectMode** is a variable that includes one of the following values. SetMultiSelectMode returns this value:
 - **true**. List applet is using multiselect mode.
 - **false**. List applet is not using multiselect mode.

For example, the following code determines whether or not the list applet that the appletIndex identifies is using multiselect mode. This code resides in the notifyobject.js file:

```
for(var appletIndex=0, len = applets.length; appletIndex < len; appletIndex++){  
    applets[appletIndex].SetMultiSelectMode(bInMultiSelectMode);
```

The physical renderer can use the AttachPMBinding method in the presentation model to bind to the SetMultiSelectMode method. The following binding allows the physical renderer to know if the list applet is in multiselect mode:

```
this.AttachPMBinding("SetMultiSelectMode", InMultiSelectMode, this);  
function InMultiSelectMode(bInMultiSelectMode){  
}
```

Presentation Model Class for Menus

This topic describes the methods that Siebel Open UI uses with the presentation models that it uses to display menus. It includes the following information:

- [Properties of the Presentation Model for Menus on page 459](#)
- [GetMenuPS Method on page 459](#)
- [OnMenuInvoke Method on page 459](#)
- [ProcessMenuCommand Method on page 460](#)
- [ShowMenu Method on page 460](#)

Properties of the Presentation Model for Menus

Table 30 describes the properties of the presentation model that Siebel Open UI uses for menus.

Table 30. Properties of the Presentation Model for Menus

| Property | Description |
|---------------|---|
| GetObjectType | Returns a string that describes object information. |
| GetRepstrName | |
| GetUIName | |
| GetId | Returns a string that describes the identifier of the menu object. Siebel Open UI gets this value from the parent menu of this menu object. |
| GetLabel | Returns a string that describes the label of the menu object. Siebel Open UI gets this value from the parent menu of this menu object. |

GetMenuPS Method

The GetMenuPS method returns a property set that includes information about a menu and the menu items that this menu contains. It uses the following syntax:

```
GetMenuPS()
```

It includes no arguments.

For example:

```
var menuPS = this.ExecuteMethod("GetMenuPS");
```

The following example includes a typical property set that the GetMenuPS method returns:

```
chiIdArray
[0]
- chiIdArray
- propArray
  - Caption : "Undo Record [Ctrl+U]"
  - Command : "*Browser Applet* *UndoRecord*SIS Account List Applet* "
  - Enabled : [True|False]
  - Type : "Command\|Separator"
```

OnMenuInvoke Method

The OnMenuInvoke method creates a menu. It returns nothing. It uses the following syntax:

```
OnMenuInvoke(consts.get("APPLET_NAME"))
```

The applicationcontext.js file includes the following code:

```
activeApplet.GetMenu().OnMenuInvoke(consts.get("APPLET_NAME"))
```

You can use the following code:

```
this.ExecuteMethod("OnMenuInvoke", consts.get("APPLET_NAME"),  
consts.get("SWE_PREPARE_APPLET_MENU"), consts.get("SWE_MENU_APPLET"), true);
```

ProcessMenuCommand Method

The ProcessMenuCommand method runs when the user chooses a menu item. It returns nothing. It uses the following syntax:

```
this.ExecuteMethod("ProcessMenuCommand", menuItemCommand);
```

It includes no arguments.

ShowMenu Method

The ShowMenu method displays a menu. It exists only for binding purposes. It makes sure Siebel Open UI finishes all processing related to the menu property set. It returns nothing. It uses the following syntax:

```
this.AttachPMBinding("ShowMenu", ShowMenuUI, this);  
function ShowMenuUI () {  
    // Include here code that displays the menu control.  
}
```

It includes no arguments.

Siebel Open UI finishes running the ShowMenu method in the proxy, and then immediately runs the ShowMenuUI method.

You must not configure Siebel Open UI to call the ShowMenu method from an external application.

Physical Renderer Class

This topic describes the methods that Siebel Open UI uses with the PhysicalRenderer class. It includes the following information:

- [BindData Method on page 461](#)
- [BindEvents Method on page 461](#)
- [EnableControl Method on page 462](#)
- [EndLife Method on page 462](#)
- [FieldChange Method for Physical Renderers on page 463](#)
- [GetPM Method for Physical Renderers on page 463](#)
- [SetControlValue Method on page 464](#)
- [ShowUI Method on page 464](#)

BindData Method

The BindData method downloads metadata and data from the Siebel Server to the client proxy, and then binds this data to the user interface. The list columns that a list applet uses is an example of metadata, and the record set that this list applet uses is an example of data. The BindData method uses the following syntax:

```
BindData(searchData, options);
```

For example, the following code in the renderer.js file uses the BindData method:

```
this.GetSearchCtrl().BindData(searchData, options);
```

For another example, the following code gets the value for a control from the presentation model, and then binds this value to this control:

```
CustomPR.prototype.BindData = function(){
    var controlSet = pm.Get("GetControls");
    for(var controlName in controlSet){
        if(controlSet.hasOwnProperty(controlName)){
            var control = controlSet[controlName];
            // Get value for this control from presentation model and bind it to
            // the control.
        }
    }
};
```

Siebel Open UI expects the physical renderer to use the BindData method to bind data to the physical control. The BindData method also gets the initial value from the presentation model, and then attaches this value to the control.

For information about:

- Examples that use BindData, see the following topics:
 - ["Customizing the Physical Renderer to Bind Data" on page 89](#)
 - ["Customizing List Applets to Render as a Carousel without Compiling the SRF" on page 173](#)
 - ["Displaying Data from External Applications in Siebel Views" on page 253](#)
- How Siebel Open UI uses BindData, see the following topics:
 - ["Life Cycle of a Physical Renderer" on page 56](#)
 - ["Guidelines for Customizing Presentation Models" on page 97](#)

BindEvents Method

The BindEvents method binds an event. It returns nothing. It uses the following syntax:

```
BindEvents(this.GetProxy().GetControls());
```

For example, the following code in the renderer.js file uses the BindEvents method:

```
this.GetConcreteRenderer().BindEvents(this.GetProxy().GetControls());
```

For another example, the following code binds a resize event:

```
CustomPR.prototype.BindEvents = function(){
    var controlSet = controls || this.GetPM().Get("GetControls");
    for(var control in controlSet){
        if(controlSet.hasOwnProperty(control)){
            // Bind for each control as required.
        }
    }
    // Resize event
    $(window).bind("resize.CustomPR", OnResize, this);
};
function OnResize(){
}
```

Siebel Open UI expects the physical renderer to use the ShowUI method to do all possible event binding. The event can reside on an applet control or in the applet area of the DOM. This binding also applies to any custom event, such as resizing a window. For more information, see [“ShowUI Method” on page 464](#) and [“Siebel CRM Events You Can Use to Customize Siebel Open UI” on page 563](#).

For information about how Siebel Open UI uses BindEvents, see the following topics:

- [“Life Cycle of a Physical Renderer” on page 56](#)
- [“Guidelines for Customizing Presentation Models” on page 97](#)
- [“Life Cycle Flows of User Interface Elements” on page 523](#)

EnableControl Method

The EnableControl method enables a control. It uses the following syntax:

```
EnableControl (control_name)
```

where:

- *control_name* identifies the name of the control that EnableControl enables.

EndLife Method

The EndLife method ends an event. It returns nothing. It uses the following syntax:

```
EndLife()
```

It includes the following arguments:

```
CustomPR.prototype.EndLife = function(){
    $(object).unbind ("event.CustomPR");
};
```

where:

- *object* identifies the object where the event runs.
- *event* identifies the name of an event.

It is recommended that you configure Siebel Open UI to end the life of any event that it no longer requires. This configuration makes sure an event handler does not continue to exist even if no object references it. For example, assume you attached a resize event on a window, and then Siebel Open UI finished running this event. The following code ends the resize event on the window object:

```
CustomPR.prototype.EndLife = function(){
    $(window).unbind("resize.CustomPR");
};
```

For information about how Siebel Open UI uses EndLife, see the following topics:

- [“Life Cycle of a Physical Renderer” on page 56](#)
- [“Guidelines for Customizing Presentation Models” on page 97](#)
- [“Using Methods with the Base Physical Renderer Class” on page 115](#)

FieldChange Method for Physical Renderers

The FieldChange method that Siebel Open UI uses with physical renderers modifies the value of a field. It returns nothing. It uses the following syntax. You add this code to the constructor method in the physical renderer:

```
this.GetPM().AttachPMBinding("FieldChange", this.SetControlValue, {scope: this})
```

It includes no arguments.

For example, you can add the following code to the constructor method that resides in the physical renderer:

```
this.GetPM().AttachPMBinding("FieldChange", this.SetControlValue, {scope: this})
);
```

This code adds the following code to the BinderMethod that resides in the physical renderer:

```
CustomPR.prototype.SetControlValue = function(control, value){
};
```

Siebel Open UI finishes running the FieldChange method, and then calls the SetControlValue method that sets the value in the physical instance of the control.

For more information, see [“AttachPMBinding Method” on page 428](#).

For information about the FieldChange method that Siebel Open UI uses with presentation models, including examples that use FieldChange, see [“FieldChange Method for Presentation Models” on page 440](#).

GetPM Method for Physical Renderers

The GetPM method returns a presentation model instance. It uses the following syntax:

```
GetPM()
```

It includes no arguments.

For example, the jqmlistrenderer.js file includes the following code:

```
var listOfColumns = this.GetPM().Get("ListOfColumns");
```

For information about:

- Examples that use GetPM, see the following topics:
 - ["Customizing the Physical Renderer to Render List Applets" on page 86](#)
 - ["Customizing the Physical Renderer to Bind Events" on page 88](#)
 - ["Customizing the Physical Renderer to Refresh the Recycle Bin" on page 89](#)
 - ["Calling Methods for Applets" on page 111](#)
 - ["Refreshing Custom Events" on page 118](#)
 - ["Text Copy of Code That Does a Partial Refresh for the Physical Renderer" on page 162](#)
 - ["Customizing List Applets to Render as a Carousel without Compiling the SRF" on page 173](#)
 - ["Adding Custom User Preferences to Applets" on page 196](#)
- How Siebel Open UI uses GetPM, see the following topics:
 - ["Life Cycle of a Physical Renderer" on page 56](#)
 - ["Using Methods with the Base Physical Renderer Class" on page 115](#)
- The GetPM method that Siebel Open UI uses for components, see ["GetPM Method for Components" on page 498](#).

SetControlValue Method

The SetControlValue method sets the value for the control that Siebel Open UI sends as an argument.

For an example that uses SetControlValue, see ["FieldChange Method for Physical Renderers" on page 463](#).

For more information about using this method, see ["Life Cycle Flows of User Interface Elements" on page 523](#).

ShowUI Method

The ShowUI method displays the physical control that corresponds to an applet control. It returns nothing. It uses the following syntax:

```
ShowUI ()
```

It includes no arguments.

For example:

```
CustomPR.prototype.ShowUI = function(){  
    var controlSet = this.GetPM().Get("GetControls");  
    for(var control in controlSet){  
        if(controlSet.hasOwnProperty(control)){  
            // Display each control, as required.  
        }  
    }  
}
```



```
    }  
  }  
};
```

A physical renderer must provide a definition for each of the following methods:

- ShowUI
- BindEvents
- BindData

It can do this definition in each of these methods or in a *superclass*, which is a parent class of the class that the method references.

For information about:

- Examples that use ShowUI, see the following topics:
 - ["Customizing the Physical Renderer to Render List Applets" on page 86](#)
 - ["Customizing List Applets to Render as a Carousel without Compiling the SRF" on page 173](#)
 - ["Adding Custom User Preferences to Applets" on page 196](#)
- How Siebel Open UI uses ShowUI, see the following topics:
 - ["Life Cycle of a Physical Renderer" on page 56](#)
 - ["Using Methods with the Base Physical Renderer Class" on page 115](#)
 - ["BindEvents Method" on page 461](#)

Business Component Class

Siebel Open UI defines the Business Component class in the component.js file. You can use the Setup method with this class. For more information, see ["Setup Method for Presentation Models" on page 433](#).

Applet Class

This topic describes the methods that Siebel Open UI uses with the Applet class. It includes the following information:

- [GetControls Method on page 466](#)
- [GetName Method for Applets on page 466](#)
- [GetRecordSet Method on page 466](#)
- [GetSelection Method on page 466](#)

Siebel Open UI defines this class in the applet.js file.

GetControls Method

The GetControls method returns the set of controls that the current applet uses. It returns this set as an object. It uses the following syntax:

```
GetControl s()
```

It includes no arguments.

For an example that uses the GetControls method, see [“Customizing Methods in the Presentation Model to Store Field Values” on page 81](#).

GetName Method for Applets

The GetName method that Siebel Open UI uses for applets returns the name of the current applet. It returns this name in a string. It uses the following syntax:

```
GetName()
```

It includes no arguments.

For information about the GetName method that Siebel Open UI uses for other classes, see [“GetName Method for Applet Controls” on page 470](#) see [“GetName Method for Application Models” on page 482](#).

GetRecordSet Method

The GetRecordSet method returns the current set of records that Siebel Open UI displays in the current applet. It returns these records in an array. It uses the following syntax:

```
GetRecordSet()
```

It includes no arguments.

GetSelection Method

The GetSelection method returns the index of the active row of the current record set. It returns this index as a number. It uses the following syntax:

```
GetSel ecti on()
```

It includes no arguments.

Applet Control Class

This topic describes the methods that Siebel Open UI uses with the Applet Control class. It includes the following information:

- [GetCaseSensitive Method on page 467](#)
- [GetDisabledBmp Method on page 468](#)
- [GetDisplayName Method on page 468](#)
- [GetDispMode Method on page 468](#)

- [GetEDEnabled Method on page 468](#)
- [GetEnabledBmp Method on page 469](#)
- [GetFieldName Method on page 469](#)
- [GetHeight Method on page 469](#)
- [GetInputName Method on page 470](#)
- [GetJustification Method on page 470](#)
- [GetMaxSize Method on page 470](#)
- [GetMethodNames Method on page 470](#)
- [GetName Method for Applet Controls on page 470](#)
- [GetPMPropSet Method on page 471](#)
- [GetPopupHeight Method on page 471](#)
- [GetPopupType Method on page 471](#)
- [GetPopupWidth Method on page 472](#)
- [GetPrompt Method on page 472](#)
- [GetUIType Method on page 473](#)
- [GetWidth Method on page 473](#)
- [IsBoundedPick Method on page 473](#)
- [IsCalc Method on page 473](#)
- [IsDynamic Method on page 473](#)
- [IsEditEnabled Method on page 474](#)
- [IsSortable Method on page 474](#)
- [NewRecord Method on page 474](#)
- [NotifyNewData Method on page 475](#)
- [SetIndex Method on page 475](#)

Each applet control references the Applet Control class. Siebel Open UI stores this class in the `appletcontrol.js` file.

GetCaseSensitive Method

The `GetCaseSensitive` method determines whether or not a control is case sensitive. It returns one of the following values:

- **1.** The control is case sensitive.
- **0.** The control is not case sensitive.

It uses the following syntax:

```
GetCaseSensitive()
```

It includes no arguments.

For example:

```
if (control.GetCaseSensitive() === "1"){  
  // This is the account control.  
  alert ("Make sure you use the correct case.");  
}
```

GetDisabledBmp Method

The GetDisabledBmp method returns the image source configured for a control if the control is disabled. It returns one of the following values depending on whether or not the image exists:

- **Exists.** Returns a string that contains the path to folder that contains the image.
- **Does not exist.** Returns nothing.

It uses the following syntax:

```
GetDisabledBmp()
```

It includes no arguments.

GetDisplayName Method

The GetDisplayName method returns the display name of a control. It returns this name in a string. It uses the following syntax:

```
GetDisplayName()
```

It includes no arguments.

For example:

```
if (control.GetDisplayName () === "Account Name"){  
  // This is the account control.  
  alert ("You are leaving Account. This will trigger an immediate post change.");  
}
```

GetDispMode Method

The GetDispMode method returns the display mode of a control. It returns this name in a string. It uses the following syntax:

```
GetDispMode()
```

It includes no arguments.

GetEDEnabled Method

The GetEDEnabled method determines whether or not an Effective Dating (ED) control is enabled. It returns one of the following values:

- **True.** Effective Dating control is enabled.

- **False.** Effective Dating control is not enabled.

It uses the following syntax:

```
GetEDEnabled()
```

It includes no arguments.

GetEnabledBmp Method

The GetEnabledBmp method determines whether or not an image source is configured for a control, whether or not this image source exists, and whether or not this control is enabled. It returns one of the following values depending on whether or not the image exists:

- **Exists.** It returns a string that contains the path to folder that contains the image.
- **Does not exist.** It returns nothing.

It uses the following syntax:

```
GetEnabledBmp()
```

- It includes no arguments.

GetFieldName Method

The GetFieldName method returns a string that includes the name of the field where a control is configured. It uses the following syntax:

```
GetFieldName()
```

It includes no arguments.

For examples that use GetFieldName, see [Customizing Methods in the Presentation Model to Store Field Values on page 81](#) and [CanNavigate Method on page 438](#).

GetHeight Method

The GetHeight method returns a string that includes the height of a control, in pixels. It uses the following syntax:

```
GetHeight()
```

It includes no arguments.

GetIndex Method

The GetIndex method returns the index of a control. This index identifies the control position in the applet. It uses the following syntax:

```
GetIndex()
```

It includes no arguments.

GetInputName Method

The GetInputName method returns a string that includes the HTML Name attribute of a control. It uses the following syntax:

```
GetInputName()
```

It includes no arguments.

For examples that use the GetInputName method, see the following topics:

- ["Text Copy of Code That Does a Partial Refresh for the Physical Renderer" on page 162](#)
- ["GetPopupType Method" on page 471](#)
- ["GetPrompt Method" on page 472](#)

GetJustification Method

The GetJustification method returns a string that indicates the text justification. It uses the following syntax:

```
GetJustification()
```

It includes no arguments.

For an example that uses the GetJustification method, see ["LookupStringCache Method" on page 487](#).

GetMaxSize Method

The GetMaxSize method returns the maximum number of characters that the user can enter into a control. It uses the following syntax:

```
GetMaxSize()
```

It includes no arguments.

GetMethodName Method

The GetMethodName method returns a string that includes the name of a method that is configured on a control. It uses the following syntax:

```
GetMethodName()
```

It includes no arguments.

For an example that uses the GetMethodName method, see ["CanInvokeMethod Method for Presentation Models" on page 437](#).

GetName Method for Applet Controls

The GetName method that Siebel Open UI uses for applet controls returns the name of an applet control. It returns this name in a string. It uses the following syntax:

```
GetName()
```

It includes no arguments.

The following example uses the GetName method:

```
if (control.GetName() === "Account"){  
  // This is the account control.  
  alert ("You are leaving Account. This will trigger an immediate post change");  
  ...  
}
```

For other examples that use the GetName method, see the following topics:

- [“Customizing the Presentation Model to Identify the Records to Delete” on page 66](#)
- [“Overriding Predefined Methods in Presentation Models” on page 74](#)
- [“Text Copy of Code That Does a Partial Refresh for the Presentation Model” on page 161](#)
- [“CanNavigate Method” on page 438](#)
- [“CanUpdate Method” on page 439](#)
- [“IsPrivateField Method” on page 444](#)
- [“CellChange Method” on page 454](#)

For information about the GetName method that Siebel Open UI uses for other classes, see [“GetName Method for Applets” on page 466](#) see [“GetName Method for Application Models” on page 482](#).

GetPMPropSet Method

The GetPMPropSet method gets the property set for a control. It uses the following syntax:

```
control.GetPMPropSet(consts.get("SWE_CTRL_PM_PS"))
```

To view an example that uses this method, see [“Customizing Control User Properties for Presentation Models” on page 108](#).

GetPopupHeight Method

The GetPopupHeight method returns a string that includes one of the following values:

- The height of the popup that is associated with a control, in pixels.
- Nothing if Siebel Open UI does not associate a popup dialog box with the control.

It uses the following syntax:

```
GetPopupHeight()
```

It includes no arguments.

For an example that uses the GetPopupHeight method, see [“GetPopupType Method” on page 471](#).

GetPopupType Method

The GetPopupType method identifies the type of popup object that Siebel Open UI associates with a control. It returns a string that includes one of the following values:

- Pick. Identifies a bounded pick list.
- Mvg. Identifies a multivalue group.
- Nothing if Siebel Open UI does not associate a popup dialog box with the control.

It uses the following syntax:

`GetPopupType()`

It includes no arguments.

The following example uses the `GetPopupType` method to make sure sufficient space exists to display the popup:

```
if (control.GetPoupType != "Pick"){
    // There's a Pick defined on this control.
    var pHeight = control.GetPopupHeight();
    var pWidth = control.GetPopupWidth();
    if (pHeight > "60" || pWidth > "200"){
        // The pop does not fit in the mobile screen, so we will disable this popup.
        var htmlName = control.GetInputName();
        // Set the control into readonly mode.
        $("[name=" + htmlName + "]").attr('readonly', true);
    }
}
```

GetPopupWidth Method

The `GetPopupWidth` method returns a string that includes one of the following values:

- The width of the popup that is associated with a control, in pixels.
- Nothing if Siebel Open UI does not associate a popup dialog box with the control.

It uses the following syntax:

`GetPopupWidth()`

It includes no arguments.

For an example that uses the `GetPopupWidth` method, see [“GetPopupType Method” on page 471](#).

GetPrompt Method

The `GetPrompt` method returns a string that includes the prompt text that Siebel Open UI displays with a control. It uses the following syntax:

`GetPrompt()`

It includes no arguments.

The following example includes the `GetPrompt` method:


```
// Alert the user when he lands in the control
if (document.getActiveElement === control.GetInputName()){
    alert (SiebelApp.S_App.LookupStringCache(control.GetPrompt()));
}
```

GetUIType Method

The GetUIType method returns a string that identifies the type of control. For example, multivalue group, picklist, calculator, and so on. It uses the following syntax:

```
GetUIType()
```

It includes no arguments.

GetWidth Method

The GetWidth method returns a string that includes the width of a control, in pixels. It uses the following syntax:

```
GetWidth()
```

It includes no arguments.

IsBoundedPick Method

The IsBoundedPick method returns one of the following values:

- **true.** The field is a bounded picklist.
- **false.** The field is not a bounded picklist.

It uses the following syntax:

```
IsBoundedPick()
```

It includes no arguments.

IsCalc Method

The IsCalc method returns one of the following values:

- **true.** The field is a calculated field.
- **false.** The field is not a calculated field.

It uses the following syntax:

```
IsCalc()
```

It includes no arguments.

IsDynamic Method

The IsDynamic method returns one of the following values:

- **true.** The control is a dynamic control.
- **false.** The control is not a dynamic control.

It uses the following syntax:

```
IsDynamic()
```

It includes no arguments.

IsEditEnabled Method

The IsEditEnabled method returns one of the following values:

- **true.** The control is editable.
- **false.** The control is not editable.

It uses the following syntax:

```
IsEditable()
```

It includes no arguments.

IsSortable Method

The IsSortable method returns one of the following values:

- **true.** The control is sortable.
- **false.** The control is not sortable.

It uses the following syntax:

```
IsSortable()
```

It includes no arguments.

NewRecord Method

The NewRecord method initializes a new record that Siebel Open UI adds to the database that resides on the Siebel Server. It uses the following syntax:

```
BusComp.prototype.NewRecord = function (blInsertBefore, blInternal, pldValue) {}
```

where:

- **blInsertBefore** can contain one of the following values:
 - **true.** Specifies to insert the record before the current record.
 - **false.** Specifies to insert the record after the current record.
- **blInternal** can contain one of the following values:
 - **true.** Configures the object manager to not call the CanInsert method to determine whether or not the insert is valid. Configures Siebel Open UI to not send a postevent notification. You can use true only if specialized business component code calls the NewRecord method.

- **false.** Configures the object manager to call the CanInsert method to determine whether or not the insert is valid. Configures Siebel Open UI to send a postevent notification.
- **pldValue** contains the value that Siebel Open UI uses as the Id for the new record. You can specify a value for pldValue to create a new record with a row Id that you specify. If you do not specify pldValue, or if it contains no value, then Siebel Open UI automatically creates a new value for the Id.

For examples that use the NewRecord method, see the following topics:

- [“Attaching an Event Handler to a Presentation Model” on page 78](#)
- [“Calling Methods for Applets” on page 111](#)
- [“Allowing Users to Return Parts” on page 378](#)
- [“SetMultipleFieldValues Method” on page 404](#)
- [“UndoRecord Method” on page 406](#)
- [“AttachPostProxyExecuteBinding Method” on page 429](#)

Note the following usage:

- NewRecord can initialize a new record, and it can also initialize a new record that includes an association with a parent record.
- You can configure Siebel Open UI to override the NewRecord method.
- The NewRecord method returns an object that includes an error code and a return value. For more information, see [“Configuring Error Messages for Disconnected Clients” on page 369](#) and [“SetErrorMsg Method” on page 420](#).
- If you use NewRecord in a Siebel Mobile disconnected environment, then NewRecord adds the record to the local database instead of the database that resides on the Siebel Server.

NotifyNewData Method

The NotifyNewData method sends an event notification to the client when Siebel Open UI modifies the value of a field. It returns nothing. It uses the following syntax:

```
BusComp.prototype.NotifyNewData = function (field_name) {}
```

where:

- *field_name* identifies the name of the field that Siebel Open UI modified.

You can use the NotifyNewData method to make sure Siebel Open UI synchronizes the modified field values between different applets that reside in the same client or that reside in different clients. NotifyNewData also notifies other fields that reference this field.

You can configure Siebel Open UI to override the NotifyNewData method.

SetIndex Method

The SetIndex method sets the index of a control. This index identifies the control position in the applet. The SetIndex method returns nothing. It uses the following syntax:

`SetIndex(value)`

where:

- *value* specifies the number to set for the index.

The following example uses the `SetIndex` method:

```
//listOfControls that contains an object of all the controls in the applet
var listOfControls = <AppletPM>.Get("GetControls");
var accountControl = listOfControls["Account"];
var accountIndex = listOfControls["Account"].GetIndex();
var revenueControl = listOfControls["Revenue"];
var revenueIndex = listOfControls["Revenue"].GetIndex();
// Now we can swap the indices and effectively the tabbing order too.
accountControl.SetIndex (revenueIndex);
revenueControl.SetIndex (accountIndex);
```

JQ Grid Renderer Class for Applets

This topic describes the methods that Siebel Open UI uses with the `JQGridRenderer` class. It includes the following information:

- [OnControlBlur Method on page 476](#)
- [OnControlMvg Method on page 477](#)
- [OnControlPick Method on page 477](#)
- [OnPagination Method on page 477](#)
- [OnRowSelect Method on page 477](#)

Siebel Open UI uses this class to render an applet as a form.

OnControlBlur Method

The `OnControlBlur` method handles an onblur event for a control that resides in a form applet. It uses the following syntax:

`OnControlBlur(arguments)`

where:

- *arguments* can include the following:
 - rowid
 - cellname
 - value
 - iRow
 - iCol

For information about the OnCtrlBlur method that Siebel Open UI uses with the presentation model for list applets, see [OnCtrlBlur Method on page 456](#).

OnControlMvg Method

The OnControlMvg method handles a multivalue group for a control that resides in a form applet. It uses the following syntax:

```
OnControlMvg(column_name)
```

where:

- *column_name* identifies the column that includes the multivalue group.

OnControlPick Method

The OnControlPick method handles a picklist for a control that resides in a form applet. It uses the following syntax:

```
OnControlPick(column_name)
```

where:

- *column_name* identifies the column that includes the picklist.

OnPagination Method

The OnPagination method handles a pagination that occurs in a form applet. It uses the following syntax:

```
OnPagination(title)
```

where:

- *title* identifies the title of the page.

OnRowSelect Method

The OnRowSelect method handles a row click. It runs if the user clicks a row. It starts the PositionOnRow that updates the proxy business component. It uses the following syntax:

```
OnRowSelect(rowId)
```

where:

- *rowId* identifies the row that the user clicked.

Business Service Class

This topic describes the method that Siebel Open UI uses with the Business Service class.

InvokeMethod Method for Business Services

The InvokeMethod method that Siebel Open UI uses for business services calls a method that resides in the proxy instance of a business service. It returns the name of the property set that this business service calls. It uses the same syntax and arguments as the InvokeMethod method that Siebel Open UI uses for application models. For more information, see [“InvokeMethod Method for Application Models” on page 485](#).

Siebel Open UI uses the GetService method of the application model class to create the method that InvokeMethod calls. For example, assume you must configure Siebel Open UI to call a business service from custom code that resides on the client, and that this code does not bind an applet control that resides in the repository to a business service. You can use InvokeMethod to call a business service method that a business service instance contains.

Assume you must configure Siebel Open UI to call the following business service:

Task UI Service (SWE)

The following code calls a business service method that a business service instance contains:

```
var service = SiebelApp.S_App.GetService(consts.get("NAME_TASKUI SVC"));
```

For more information, see [“GetService Method” on page 483](#).

The following code calls the GoToInbox method:

```
if(service){outPS = service.InvokeMethod("GoToInbox", inPS, true);}
```

Application Model Class

This topic describes the methods that Siebel Open UI uses with the Application Model class. It includes the following information:

- [CanInvokeMethod Method for Application Models on page 479](#)
- [ClearMainView Method on page 479](#)
- [GenerateSrvrReq Method on page 479](#)
- [GetAccessibilityEnhanced Method on page 480](#)
- [GetActiveBusObj Method on page 480](#)
- [GetActiveView Method on page 480](#)
- [GetAppletControllInstance Method on page 481](#)
- [GetAppTitle Method on page 482](#)
- [GetDirection Method on page 482](#)
- [GetName Method for Application Models on page 482](#)
- [GetPageURL Method on page 482](#)
- [GetProfileAttr Method on page 483](#)
- [GetService Method on page 483](#)

- [GotoView Method on page 484](#)
- [InvokeMethod Method for Application Models on page 485](#)
- [IsExtendedKeyBoard Method on page 486](#)
- [IsMobileApplication Method on page 487](#)
- [LogOff Method on page 487](#)
- [LookupStringCache Method on page 487](#)
- [RemoveService Method on page 488](#)
- [NewProperty Set Method on page 488](#)
- [SetDiscardUserState Method on page 488](#)

CanInvokeMethod Method for Application Models

The CanInvokeMethod method that Siebel Open UI uses for application models determines whether or not Siebel Open UI can invoke a method. It uses the same syntax as the CanInvokeMethod method that Siebel Open UI uses for presentation models. For more information, see [“CanInvokeMethod Method for Presentation Models” on page 437](#).

ClearMainView Method

The ClearMainView method removes values for the following items:

- The view
- All child objects of the view, such as applets and controls
- The business object that the view references
- Child objects of the business object that the view references, such as business components and business component fields

ClearMainView uses the following syntax:

```
ClearMainView()
```

ClearMainView only removes values for objects that reside in the client. It does not visually destroy these objects.

If the user attempts to use an object that ClearMainView has cleared, then Siebel Open UI might not work as expected.

GenerateSrvrReq Method

The GenerateSrvrReq method creates a request string that Siebel Open UI sends to the Siebel Server according to the current context of the application. It returns a string that includes a description of the full request. It uses the following syntax:

```
GenerateSrvrReq (command)
```

where:

■ *command* is a string that identifies the name of the command that Siebel Open UI must request.

For example:

```
var request = SiebelApp.S_App.GenerateSrvrReq("LogOff");
```

In this example, the return value includes a string that contains the following information:

```
http(s)://server_name/callcenter_enu/
start.swe?SWECmd=LogOff&SWEKeepContext=1&SWERPC=1&SRN=L8ct6oeEsPA3Cj7pF6spebyCLm2m
VGpBOD0tqGMcfIcb&SWEC=18&SWEActiveApplet=Client Active Applet&SWEActiveView=Client
Active View
```

GetAccessibilityEnhanced Method

The GetAccessibilityEnhanced method determines whether or not accessibility is enabled. It returns a string that includes one of the following values:

- **true.** Accessibility is enabled.
- **false.** Accessibility is not enabled.

It uses the following syntax:

```
GetAccessibilityEnhanced()
```

It includes no arguments.

GetActiveBusObj Method

The GetActiveBusObj method returns the name of the business object that is currently active in the client. It uses the following syntax:

```
GetActiveBusObj()
```

It includes no arguments.

For example:

```
var busObj = SiebelApp.S_App.GetActiveBusObj();
var busComp = busObj.GetBusCompByName("MyBusComp");
var canUpdate = busComp.CanUpdate();
if (canUpdate){
    ...
}
```

GetActiveView Method

The GetActiveView method returns the name of the view that is currently active in the client. It uses the following syntax:

```
GetActiveView()
```

It includes no arguments.

For example:

```
var view = Siebel App. S_App. GetActiveView();
var applet = view. GetActiveApplet();
var canUpdate = applet. CanUpdate();
if (canUpdate){
    ...
}
```

For more examples that use the `GetActiveView` method, see the following topics:

- [“Creating Components” on page 119](#)
- [“Using JavaScript to Customize the Browser Tab Label” on page 151](#)
- [“Displaying Data from External Applications in Siebel Views” on page 253](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Return Parts” on page 378](#)
- [“Allowing Users to Set the Activity Status” on page 384](#)
- [“Name Method for Applets” on page 389](#)

GetAppletControlInstance Method

The `GetAppletControlInstance` method creates a control. It returns the name of the control that it creates. It uses the following syntax:

```
GetAppletControlInstance (name, uiType, displayName, width, height)
```

where:

- *name* is a string that contains the name that Siebel Open UI assigns to the control.
- *uiType* is a string that identifies the type of the control. For more information, see *Siebel Object Types Reference*.
- *displayName* is a string that contains the name of the control that Siebel Open UI displays in the client.
- *width* is a string that contains a number that specifies the width of the control, in pixels.
- *height* is a string that contains a number that specifies the height of the control, in pixels.

For example:

```
var myControl = Siebel App. S_App. GetAppletControlInstance (
    "MyDropDown",
    constants.get("SWE_CTRL_COMBOBOX"),
    "I want this to appear on the screen",
    "50",
    "20");
```

For another example that uses the `GetAppletControlInstance` method, see [“Customizing the Setup Logic of the Presentation Model” on page 64](#).

GetAppTitle Method

The GetAppTitle method returns the title of the current Siebel application. It returns this title in a string. It uses the following syntax:

```
GetAppTi tle()
```

It includes no arguments.

For example:

```
var appTi tle = Siebel App. S_App. GetAppTi tle();  
if (appTi tle === "Siebel Cal l Center"){  
    ...  
}
```

GetDirection Method

The GetDirection method determines the direction that Siebel Open UI uses to display text. It returns one of the following values:

- **RTL.** Siebel Open UI is configured so the user reads text from right-to-left.
- **Null.** Siebel Open UI is not configured so the user reads text from right-to-left.

It uses the following syntax:

```
GetDi recti on()
```

It includes no arguments.

GetName Method for Application Models

The GetName method that Siebel Open UI uses for application models returns the name of the current Siebel application. For example, Siebel Call Center. It returns this name in a string. It uses the following syntax:

```
GetName()
```

It includes no arguments.

For example:

```
acti veVi ew. ExecuteFrame (acti veAppl et. GetName(), [{fi el d: thi s. Get("SearchFi el d"),  
    val ue: thi s. Get("SearchVal ue")}])
```

For information about the GetName method that Siebel Open UI uses for other classes, see [“GetName Method for Applets” on page 466](#) see [“GetName Method for Applet Controls” on page 470](#).

GetPageURL Method

The GetPageURL method returns the URL that the Siebel application uses. It returns this value without a query string. For example, it can return the following value:

```
http: //computer_name. exampl e. com/start. swe
```

It uses the following syntax:

```
GetPageURL()
```

It includes no arguments.

For example:

```
final url = Siebel App. S_App. GetPageURL() + strURL. split("start.swe")[1];
```

GetProfileAttr Method

The GetProfileAttr method returns the value of a user profile attribute. It uses the following syntax:

```
GetProfileAttr (attribute_name)
```

where:

■ *attribute_name* is a string that includes the name of an attribute.

For examples that use the GetProfileAttr method, see [“Adding Custom Manifest Expressions” on page 143](#) and [“Configuring Siebel Open UI to Use Different Web Templates According to the Applet Mode” on page 190](#).

GetService Method

The GetService method creates a business service instance that allows Siebel Open UI to call a business service method that this business service instance contains. It returns the business service name. It uses the following syntax:

```
Siebel App. S_App. GetService("name");
```

where:

■ *name* is a string that identifies the name of the business service that GetService calls when it creates the business service instance.

For example, assume you must configure Siebel Open UI to call a business service from custom code that resides on the client, and that this code does not bind an applet control that resides in the repository to a business service. You can use the GetService method to create a business service instance that Siebel Open UI can use to call a business service method that this business service contains.

Assume you must configure Siebel Open UI to call the following business service:

Task UI Service (SWE)

The following code creates an instance of this business service:

```
var service = Siebel App. S_App. GetService("Task UI Service (SWE)");
```

You can configure Siebel Open UI to call a business service method that this business service contains after this instance is available. For example, the following code calls the GoToInbox method that the Task UI Service (SWE) business service contains:

```
if(service){outPS = service.InvokeMethod("GoToInbox", inPS, true);}
```

For more examples that use GetService, see the following topics:

- [“Calling Methods for Business Services” on page 112](#)
- [“Allowing Users to Interact with Clients During Business Service Calls” on page 120](#)
- [“RemoveService Method” on page 488](#)

For information about Siebel Open UI uses GetService with InvokeMethod, see [“InvokeMethod Method for Business Services” on page 478](#).

GotoView Method

The GotoView method navigates the user to a view in the client. It uses the following syntax:

```
Siebel App. S_App. GotoView(view, viewId, strURL, strTarget);
```

where:

- view is an object that contains the name of the view. It is required. Other arguments are optional.
- viewId is an object that contains the Id of the view.
- strURL is an object that contains a string that Siebel Open UI sends as part of the GotoView method. This string must use the HTTP query string format that Siebel CRM requires. For example:

```
"SWEParam1=valueForParam1&SWEParam2=valueForParam2"
```

- strTarget is an object that contains the string target.

For example, assume view contains a value of Account List View. The following code navigates the user to this view:

```
Siebel App. S_App. GotoView(view, viewId, strURL, strTarget);
```

For more examples that use the GotoView method, see the following topics:

- [“SetDiscardUserState Method” on page 488](#)
- [“Displaying Siebel Portlets In External Applications” on page 261](#)
- [“Alternatives to Configuring the URL” on page 269](#)
- [“Using iFrame Gadgets to Display Siebel CRM Applets in External Applications” on page 272](#)

For more information about using this method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

Work That Siebel Open UI Does When it Runs the GotoView Method

Siebel Open UI does the following work when it runs the GotoView method:

- 1 Sets the cursor state to busy.
- 2 Runs any required validation steps. If a validation fails in the client, then Siebel Open UI returns a value of false and exits the GotoView method. Implicit Commit is an example of a validation.
- 3 Adds default arguments.
- 4 Sends a request to the Siebel Server.

- 5 Navigates the user to the view that view specifies.

InvokeMethod Method for Application Models

The InvokeMethod method that Siebel Open UI uses for application models calls a method. It returns a value from the method that it calls. It uses the following syntax:

```
Siebel App. S_App. InvokeMethod("method_name", psObject, ai);
```

where:

- *method_name* identifies the name of the method that InvokeMethod calls.
- *psObject* is an object that contains a property set that InvokeMethod sends as input to the method that it calls, if required.
- *ai* is an object that contains information about how to run AJAX. For more information, see [“Values That You Can Set for the AI Argument” on page 486](#).

For example, the following code calls the NextApplet method. This method sets the next applet as the active applet of a view:

```
Siebel App. S_App. InvokeMethod("NextApplet", psObject, ai);
```

For more examples that use the InvokeMethod method, including for Disconnected clients, see the following topics:

- [“Customizing the Presentation Model to Delete Records” on page 69](#)
- [“Attaching an Event Handler to a Presentation Model” on page 78](#)
- [“Calling Methods for Applets” on page 111](#)
- [“Allowing Users to Interact with Clients During Business Service Calls” on page 120](#)
- [“Customizing Predefined Applets” on page 363](#)
- [“Using Custom JavaScript Methods” on page 364](#)
- [“Using Custom Siebel Business Services” on page 367](#)
- [“Customizing Siebel Pharma for Siebel Mobile Disconnected Clients” on page 372](#)
- [“Allowing Users to Commit Part Tracker Records” on page 376](#)
- [“Allowing Users to Return Parts” on page 378](#)

For more information about using InvokeMethod, see [“Calling Methods for Applets and Business Services” on page 111](#).

For more information about the InvokeMethod method that Siebel Open UI uses for other classes, see [“InvokeMethod Method for Presentation Models” on page 443](#) and [“InvokeMethod Method for Business Services” on page 478](#).

Values That You Can Set for the AI Argument

You can use the ai (additional input) argument to specify how Siebel Open UI runs an AJAX (Asynchronous JavaScript And XML) request. For more information about AJAX, see the Web site that describes Including Ajax Functionality in a Custom JavaServer Faces Component at <http://www.oracle.com/technetwork/java/javaee/tutorial-jsp-140089.html>.

Table 31 lists the values that you can use with the ai argument. For example, to use the async value, you use ai . async.

Table 31. Values That You Can Set for the AI Argument

| Value | Description |
|----------|---|
| async | Asynchronous. Set to one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI makes an asynchronous AJAX call. ■ false. Siebel Open UI makes a synchronous AJAX call. |
| cb | Callback. Identifies the method that Siebel Open UI calls after it receives a reply from the AJAX call. For more information, see “Coding Callback Methods” on page 375 . |
| scope | Set to the following value: <p>thi s</p> |
| errcb | Error callback. Identifies the method that Siebel Open UI calls after it receives a reply from the AJAX call if this AJAX call fails. For more information, see “Coding Callback Methods” on page 375 . |
| opdecode | Decode operation. Set to one of the following values: <ul style="list-style-type: none"> ■ true. Decode the AJAX reply. ■ false. Do not decode the AJAX reply. |
| mask | Set to one of the following values: <ul style="list-style-type: none"> ■ true. Mask the screen. ■ false. Do not mask the screen. <p>If selfbusy is true, then mask does nothing.</p> |
| selfbusy | Determines whether or not Siebel Open UI displays a busy cursor while it processes the AJAX request. Set to one of the following values: <ul style="list-style-type: none"> ■ true. Do not display a busy cursor. ■ false. Display a busy cursor. |

IsExtendedKeyBoard Method

The IsExtendedKeyBoard method determines whether or not Siebel Open UI is configured to use extended keyboard shortcuts. It returns one of the following values:

- **true**. Siebel Open UI is configured to use extended keyboard shortcuts.

- **false.** Siebel Open UI is not configured to use extended keyboard shortcuts.

It uses the following syntax:

```
IsExtendedKeyBoard()
```

It includes no arguments.

IsMobileApplication Method

The IsMobileApplication method determines whether or not Mobile is enabled for the Siebel application that is currently running in the client. It returns a string that includes one of the following values:

- **true.** Mobile is enabled.
- **false.** Mobile is not enabled.

It uses the following syntax:

```
IsMobileApplication()
```

It includes no arguments.

For an example that uses the IsMobileApplication method, see [“Determining Whether or Not Siebel Open UI Is Enabled for Siebel Mobile” on page 278](#).

LogOff Method

The LogOff method calls the Siebel Server, and then returns the Login page to the client. It uses the following syntax:

```
LogOff()
```

It includes no arguments.

LookupStringCache Method

The LookupStringCache method gets a string from the client string cache. It uses the following syntax:

```
LookupStringCache (index)
```

where:

- *index* is a number that identifies the location of a string that resides in the client string cache.

For example:

```
// Assume appletControl to be the reference of an applet control.
var justification = appletControl.GetJustification(); //Returns text justification
in index.
var stringJustification = SiebelApp.S_App.LookupStringCache (justification);
alert (justification); // Will alert "Left" or "Right"
```

For another example that uses the LookupStringCache method, see [“GetPrompt Method” on page 472](#).

NewPropertySet Method

The NewPropertySet method creates a new property set instance. It returns this instance. It uses the following syntax:

```
NewPropertySet ()
```

It includes no arguments.

For example, the following code resides in the alarm.js file:

```
var returnPropSet = App ().NewPropertySet();
```

For more examples that use the NewPropertySet method, see [“Customizing the Presentation Model to Delete Records” on page 69](#) and [“Allowing Users to Interact with Clients During Business Service Calls” on page 120](#).

RemoveService Method

The RemoveService method removes a business service from the client. It uses the following syntax:

```
RemoveService (business_service_name)
```

where:

■ *business_service_name* identifies the name of the business service that Siebel Open UI removes.

For example, the following code removes the Task UI Service (SWE) business service:

```
var service = SiebelApp.S_App.GetService("Task UI Service (SWE)");  
// Use service  
...  
//Remove service  
if (service){
```

If you use RemoveService to remove a business service that does not exist, then Siebel Open UI might not behave as predicted.

SetDiscardUserState Method

The SetDiscardUserState method sets a property in the client that configures Siebel Open UI to not evaluate the state before it navigates to another view. It uses the following syntax:

```
SetDiscardUserState (binary)
```

where:

■ *binary* is one of the following values:

- **true.** Ignore the state before doing navigation. Siebel Open UI applies this logic for all potential states, such as a commit is pending, Siebel Open UI is currently opening a dialog box, and so on. Siebel Open UI runs any GotoView call it receives. It loses the client state.
- **false.** Do not ignore the state before doing navigation. Do the client validation.

For example:


```
// A business condition is met that requires Siebel Open UI to automatically navigate
the user.
Siebel App. S_App. DiscardUserState (true);
// Don't care about user state - we need the navigation to occur.
Siebel App. S_App. GotoView ("MyView" . . );
// Reset
Siebel App. S_App. DiscardUserState (false);
```

Control Builder Class

Table 32 describes the methods that Siebel Open UI uses with the ControlBuilder class.

Table 32. Methods You Can Use with the SiebelAppFacade.ControlBuilder Class

| Method | Description |
|--------------|---|
| Pick | You can use the following properties of the configuration object: |
| Mvg | <ul style="list-style-type: none"> ■ target. Specifies the DOM element as a jQuery object. ■ click. Attaches a callback method. ■ scope. Specifies the scope. ■ control. Sent as an argument to the callback method that the click property specifies. For more information, see "Coding Callback Methods" on page 375. ■ className. Modifies the CSS style of the pick icon. |
| DatePick | You can use the following properties of the configuration object: |
| DateTimePick | <ul style="list-style-type: none"> ■ target. Specifies the input control DOM element as a jQuery object with a calendar icon and attaches a DatePicker event. Configures Siebel Open UI to display a dialog box that contains only date options if the user clicks the calendar icon. <p>DateTimePick does the same as DatePick except the dialog box allows the user to set the date and time.</p> <ul style="list-style-type: none"> ■ className. Identifies the class. |

Locale Object Class

This topic describes the methods that Siebel Open UI uses with the Locale Object class. It includes the following information:

- [FormattedToString Method on page 490](#)
- [GetCurrencyList Method on page 491](#)
- [GetDateFormat Method on page 491](#)

- [GetDayOfWeek Method on page 491](#)
- [GetDispCurrencyDecimal Method on page 491](#)
- [GetDispCurrencySeparator Method on page 492](#)
- [GetDispDateSeparator Method on page 492](#)
- [GetDispNumberDecimal Method on page 492](#)
- [GetDispNumberScale Method on page 492](#)
- [GetDispNumberSeparator Method on page 492](#)
- [GetDispTimeAM Method on page 493](#)
- [GetDispTimePM Method on page 493](#)
- [GetDispTimeSeparator Method on page 493](#)
- [GetExchangeRate Method on page 493](#)
- [GetFuncCurrCode Method on page 494](#)
- [GetLocalString Method on page 494](#)
- [GetMonth Method on page 494](#)
- [GetScale Method on page 495](#)
- [GetStringFromDateTime Method on page 495](#)
- [GetTimeFormat Method on page 495](#)
- [GetTimeZoneList Method on page 495](#)
- [GetTimeZoneName Method on page 496](#)
- [SetCurrencyCode Method on page 496](#)
- [SetExchDate Method on page 496](#)
- [SetScale Method on page 496](#)
- [StringToFormatted Method on page 497](#)

FormattedToString Method

The FormattedToString method removes the formatting of a string. It returns the unformatted string. It uses the following syntax:

`FormattedToString(type, value, format)`

where:

- *type* is a string that identifies the value type of the string. For example: Phone, Currency, DateTime, or Integer.
- *value* is a string that identifies the formatted value of the string.
- *format* is a string that identifies the optional format of the string.

For example:

```
Siebel App. S_App. Local eObject. FormattedToString("date", "11/05/2012", "M/D/YYYY")
```

GetCurrencyList Method

The GetCurrencyList method returns the currency list that the client computer supports. It uses the following syntax:

```
GetCurrencyList()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetCurrencyList()
```

GetDateFormat Method

The GetDateFormat method returns the date format for the locale. It uses the following syntax:

```
GetDateFormat()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetDateFormat()
```

GetDayOfWeek Method

The GetDayOfWeek method returns a string that identifies the day of the week. It uses the following syntax:

```
GetDayOfWeek(day, format)
```

where:

- *day* is a number that indicates the index of the day.
- *format* is string that specifies the day format.

For example:

```
Siebel App. S_App. Local eObject. GetDayOfWeek(20, "M/D/YYYY")
```

GetDispCurrencyDecimal Method

The GetDispCurrencyDecimal method returns the decimal point symbol that the client uses for currency, such as a period (.). It uses the following syntax:

```
GetDispCurrencyDecimal()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetDispCurrencyDecimal()
```

GetDispCurrencySeparator Method

The GetDispCurrencySeparator method returns the number separator that the currency uses to separate digits in a currency, such as a comma (,). It uses the following syntax:

```
GetDispCurrencySeparator()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocalObject.GetDispCurrencySeparator()
```

GetDispDateSeparator Method

The GetDispDateSeparator method returns the symbol that the client uses to separate the days, weeks, and months of a date. It uses the following syntax:

```
GetDispDateSeparator()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocalObject.GetDispDateSeparator()
```

GetDispNumberDecimal Method

The GetDispNumberDecimal method returns the symbol that the client uses for the decimal point. For example, a period (.). It uses the following syntax:

```
GetDispNumberDecimal()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocalObject.GetDispNumberDecimal()
```

GetDispNumberScale Method

The GetDispNumberScale method returns the number of fractional digits that the client displays. For example, 2. It uses the following syntax:

```
GetDispNumberScale()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocalObject.GetDispNumberScale()
```

GetDispNumberSeparator Method

The GetDispNumberSeparator method returns the symbol that the client uses to separate digits in a number. For example, the comma (,). It uses the following syntax:

`GetDisplayNumberSeparator()`

It includes no arguments.

For example:

`SiebelApp.S_App.LocalObject.GetDisplayNumberSeparator()`

GetDisplayTimeAM Method

The `GetDisplayTimeAM` method returns the localized string for AM. For example, AM. It uses the following syntax:

`GetDisplayTimeAM()`

It includes no arguments.

For example:

`SiebelApp.S_App.LocalObject.GetDisplayTimeAM()`

GetDisplayTimePM Method

The `GetDisplayTimePM` method returns the localized string for PM. For example, PM. It uses the following syntax:

`GetDisplayTimePM()`

It includes no arguments.

For example:

`SiebelApp.S_App.LocalObject.GetDisplayTimePM()`

GetDisplayTimeSeparator Method

The `GetDisplayTimeSeparator` method returns the symbol that the client uses to separate the parts of time. For example, the colon (:) symbol. It uses the following syntax:

`GetDisplayTimeSeparator()`

It includes no arguments.

For example:

`SiebelApp.S_App.LocalObject.GetDisplayTimeSeparator()`

GetExchangeRate Method

The `GetExchangeRate` method calculates the exchange rate between two currencies. It returns the exchange rate as a double precision, floating point number. It uses the following syntax:

`GetExchangeRate(input_value, output_value, exchange_date)`

where:

- *input_value* is a string that identifies the currency code that Siebel Open UI uses for the input value when it calculates the exchange rate.
- *output_value* is a string that identifies the currency code that Siebel Open UI uses for the output value when it calculates the exchange rate.
- *exchange_date* is a string that includes the date of the currency exchange.

For example:

```
Siebel App. S_App. Local eObject. GetExchangeRate("USD", "INR", "11/05/2012")
```

GetFuncCurrCode Method

The GetFuncCurrCode method returns the currency code that the client uses. For example, USD. It uses the following syntax:

```
GetFuncCurrCode()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetFuncCurrCode()
```

GetLocalizedString Method

The GetLocalizedString method returns the localized string of a key. It uses the following syntax:

```
GetLocal String(pStringKey : string_name : message_key)
```

where:

- *pStringKey* is a property set that includes the string key.
- *string_name* is a string that identifies the name of the localized string that GetLocalizedString gets.

For example, the following code uses the GetLocalizedString method when using Siebel Open UI with a connected client:

```
Siebel App. S_App. Local eObject. GetLocal String("IDS_SWE_LOADING_INDICATOR_TITLE")
```

For another example, the following code uses the GetLocalizedString method when using Siebel Open UI with Siebel Mobile disconnected:

```
Siebel App. S_App. OfflineLocal eObject. GetLocal String("IDS_SWE_LOADING_INDICATOR_TITLE")
```

GetMonth Method

The GetMonth method returns the month that the locale uses. It uses the following syntax:

```
GetMonth()
```

It includes no arguments.

For example:

Siebel App. S_App. Local eObject. GetMonth()

GetScale Method

The GetScale method returns the scale of the number that Siebel Open UI must display. It uses the following syntax:

GetScale()

It includes no arguments.

For example:

Siebel App. S_App. Local eObject. GetScale()

GetStringFromDateTime Method

The GetStringFromDateTime method formats a date and time string. It returns this formatted date and time in a string. It uses the following syntax:

GetStringFromDateTime(*input_date*, *input_format*, *output_format*)

where:

- *input_date* specifies the date that GetStringFromDateTime formats.
- *input_format* describes how *input_date* is formatted.
- *output_format* specifies how to format the output.

For example:

Siebel App. S_App. Local eObject. GetStringFromDateTime(2012-12-05, DD/MM/YYYY, M/D/YYYY)

GetTimeFormat Method

The GetTimeFormat method returns the time format that the locale uses. It uses the following syntax:

GetTimeFormat()

It includes no arguments.

For example:

Siebel App. S_App. Local eObject. GetTimeFormat()

GetTimeZoneList Method

The GetTimeZoneList method returns a list of time zones that the locale uses. It uses the following syntax:

GetTimeZoneList()

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetTimezoneList()
```

GetTimeZoneName Method

The GetTimeZoneName method returns the current time zone that the locale uses. It uses the following syntax:

```
GetTimeZoneName()
```

It includes no arguments.

For example:

```
Siebel App. S_App. Local eObject. GetTimeZoneName()
```

SetCurrencyCode Method

The SetCurrencyCode method sets the currency code that the locale uses. It returns nothing. It uses the following syntax:

```
SetCurrencyCode(currency_code)
```

where:

■ *currency_code* is a string that includes the currency code.

For example:

```
Siebel App. S_App. Local eObject. SetCurrencyCode("USD")
```

SetExchDate Method

The SetExchDate method sets the exchange date that the currency uses. It returns nothing. It uses the following syntax:

```
SetExchDate(exchange_date)
```

where:

■ *exchange_date* is a string that includes the exchange date.

For example:

```
Siebel App. S_App. Local eObject. SetExchDate("11/05/2012")
```

SetScale Method

The SetScale method sets the scale of the number. It returns nothing. It uses the following syntax:

```
SetScale(scale)
```

where:

■ *scale* is a string that includes the number that SetScale uses to set the scale.

For example:

```
Siebel App. S_App. Local eObject. SetScale("0")
```

StringToFormatted Method

The StringToFormatted method adds formatting characters to a string. It returns a formatted string. It uses the following syntax:

```
StringToFormatted(type, value, format)
```

The StringToFormatted method uses the same arguments that the FormattedToString method uses. For more information, see ["FormattedToString Method" on page 490](#).

For example:

```
Siebel App. S_App. Local eObject. StringToFormatted("date", "11/05/2012", "M/D/YYYY")
```

Component Class

This topic describes the methods that Siebel Open UI uses with the Component class. It includes the following information:

- ["Component Method" on page 497](#)
- ["GetChildren Method" on page 498](#)
- ["GetParent Method" on page 498](#)
- ["GetPM Method for Components" on page 498](#)
- ["GetPR Method" on page 499](#)
- ["GetSiblings Method" on page 499](#)
- ["Setup Method for Components" on page 499](#)
- ["Show Method for Components" on page 500](#)

Component Method

The Component method is a constructor that creates a component object. It returns nothing. It uses the following syntax:

```
Component()
```

It includes no arguments.

For example:

```
var cmpObj = new Siebel AppFacade. Component();
```

GetChildren Method

The GetChildren method identifies all child components that a parent component contains. It returns these child components in an array. If no child components exist, then it returns nothing. It uses the following syntax:

```
GetChildren()
```

It includes no arguments.

For example:

```
var childrenCmp = cmpObj.GetChildren();
```

where:

■ cmpObj references a component object.

GetParent Method

The GetParent method gets the parent component object. Siebel Open UI uses a tree structure to manage components. It uses this structure to identify the parent component that a query examines. It uses the following syntax:

```
GetParent()
```

It includes no arguments.

For example:

```
var parentObj = cmpObj.GetParent();
```

where:

■ cmpObj references a component object.

GetPM Method for Components

The GetPM method that Siebel Open UI uses for components returns the presentation model object that the component references. It uses the following syntax:

```
GetPM()
```

It includes no arguments.

For example:

```
var pmObj = cmpObj.GetPM();
```

where:

■ cmpObj references a component object.

If you use GetPM before Siebel Open UI runs the setup call for the component, then GetPM returns a value that indicates that Siebel Open UI has not yet defined the presentation model object that this component references.

For information about the GetPM method that Siebel Open UI uses for physical renderers, see [“GetPM Method for Physical Renderers” on page 463](#).

GetPR Method

The GetPR method returns a physical renderer object that is associated with a component. It uses the following syntax:

```
GetPR()
```

It includes no arguments.

For example:

```
var prObj = cmpObj.GetPR();
```

where:

- `cmpObj` references a component object.

Siebel Open UI defers creating the physical renderer until it calls the Show function in the component.

GetSiblings Method

The GetSiblings method returns all *siblings*. In this context, a sibling is a component that reside at same the level in the component tree structure as the component that it calls. It returns these values in an array. If no other components reside at the same level, then it returns nothing. The GetSiblings method uses the following syntax:

```
GetSiblings()
```

It includes no arguments.

For example:

```
var siblingObjs = cmpObj.GetSiblings();
```

where:

- `cmpObj` references a component object.

Setup Method for Components

The Setup method that Siebel Open UI uses with components does the basic setup for the component instance, and then prepares the presentation model that this component instance references. It calls the Setup method that resides in this presentation model. It uses the following syntax:

```
Setup(property_set)
```

where:

- `property_set` identifies a property set that Siebel Open UI passes to the presentation model that the component references.

The Component Manager calls the Setup method. It is recommended that you do not configure Siebel Open UI to directly call the setup method on any component object.

For more information about the Setup method that Siebel Open UI uses with presentation models, see [“Setup Method for Presentation Models” on page 433](#).

Show Method for Components

The Show method that Siebel Open UI uses for components shows a component. It uses the following syntax:

```
Show()
```

It includes no arguments.

Siebel Open UI uses the Component Manager to call the Show method for a component. This Show method does the following work during this call:

- If the physical renderer object does not already exist, then the Component Manager creates it.
- Calls the following methods that reside in the physical renderer:
 - ShowUI
 - BindEvents
 - BindData

For more information about how Siebel Open UI uses these methods, see [“Life Cycle of a Physical Renderer” on page 56](#).

- Calls the Show method for every component object it creates while it runs, as necessary.

In some situations, Siebel Open UI might not finish calling the Setup method if it creates the component after the Component Manager life cycle finishes. In this situation, Siebel Open UI can use the Show method to call this component to make sure that it completes this life cycle successfully. For more information, see [“Setup Method for Components” on page 499](#).

It is recommended that you not configure Siebel Open UI to make a direct call to the Show method for a component.

For more information about using the Show method, see [“Life Cycle Flows of User Interface Elements” on page 523](#).

For information about the Show method that Siebel Open UI uses for component managers, see [“Show Method for Component Managers” on page 502](#).

Component Manager Class

This topic describes the methods that Siebel Open UI uses with the Component Manager class. It includes the following information:

- [“DeleteComponent Method” on page 501](#)
- [“FindComponent Method” on page 501](#)

- [“MakeComponent Method” on page 502](#)
- [“Show Method for Component Managers” on page 502](#)

The Component Manager class manages components in Siebel Open UI. It can create or delete components and it allows you to configure Siebel Open UI to search for a component according to criteria that you specify.

DeleteComponent Method

The DeleteComponent method deletes a component from the component tree. It uses the following syntax:

```
DeleteComponent(cmpObj)
```

where:

- `cmpObj` references a component object.

For example, the following code deletes the component that `cmpObj` references:

```
SiebelAppFacade.ComponentMgr.DeleteComponent(cmpObj);
```

FindComponent Method

The FindComponent method identifies a component according to the criteria that a function specifies. It returns an array that includes component names. If it cannot identify any components, then it returns nothing. It uses the following syntax:

```
FindComponent({id : "custom_dependency_object"});
```

Finding Components According to IDs

Siebel Open UI maps the Id of the component to the name of this component. It does the same mapping when it uses the MakeComponent method to create a dependency. You can use the following code to find a component according to the component Id:

```
var cmpObj = SiebelAppFacade.ComponentMgr.FindComponent({id :  
"custom_dependency_object"});
```

Getting Parents, Siblings, and Children

If you provide a component and a relation, then the FindComponent method gets a list of components according to the component and relation that you specify. You use the following code:

```
var cmprelationship = SiebelAppFacade.ComponentMgr.FindComponent({cmp: cmpObj, rel  
: consts.get("values")});
```

where:

- `relationship` specifies a parent, sibling, or child relationship.
- `cmp` is an abbreviation for component. `cmpObj` identifies the component.
- `rel` is an abbreviation for relation. It identifies the type of relationship.

■ *values* specifies the values to get. To get a list of:

- Parents, you use SWE_CMP_REL_SIBLING
- Siblings, you use SWE_CMP_REL_SIBLING
- Children, you use SWE_CMP_REL_CHILDREN

For example, the following code gets a list of parents:

```
var cmpParent = SiebelAppFacade.ComponentMgr.FindComponent({cmp: cmpObj, rel :  
  consts.get("SWE_CMP_REL_PARENT")});
```

MakeComponent Method

The MakeComponent method creates a component. It returns nothing. It uses the following syntax:

```
SiebelAppFacade.ComponentMgr.MakeComponent(parent, psInfo, dependency);
```

where:

- *parent* identifies the parent of the component that Siebel Open UI creates. For example, a view, applet, and so on.
- *psInfo* contains property set information that identifies the name of the module that Siebel Open UI uses for the presentation model and the physical renderer. Siebel Open UI uses this property set information to create the presentation model. It also passes this property set to the setup method that it uses to set up the presentation model.
- *dependency* identifies an object that Siebel Open UI uses as a template to create the presentation model. If the presentation model must reference an applet or view, then this dependency must also reference this same applet or view. To specify the dependency for a local component, you must use an object that references the GetName method.

The MakeComponent method does the following work:

- Creates a component.
- Attaches this component to the component tree. It attaches this component at the tree level that Siebel Open UI uses for user interface objects.
- Calls the Setup method that Siebel Open UI uses to create the new component. This Setup method uses information that the psInfo argument of the MakeComponent method specifies. It uses this information to create the presentation model. For more information, see [“Setup Method for Components” on page 499](#).
- Calls the Setup method that Siebel Open UI uses for the presentation model. This method binds all objects that are involved in the life cycle that Siebel Open UI runs for the component. For more information, see [“Setup Method for Presentation Models” on page 433](#).

For an example that uses the MakeComponent method, see [“Creating Components” on page 119](#).

Show Method for Component Managers

The Show method that Siebel Open UI uses for component managers displays components. It uses the following syntax:

Show()

It includes no arguments.

The Show method that Siebel Open UI uses for component managers calls a show on the component object. This component object then calls a Show method on the physical renderer that the component references.

You can use the Show method to configure Siebel Open UI to display all components that reside in the tree that contains the component. If you must configure Siebel Open UI to display only one component, then is recommended that you use the Show method on each individual component.

For information about the Show method that Siebel Open UI uses for components, see [“Show Method for Components” on page 500](#).

Cascading Style Sheet Classes

[Table 33](#) lists the classes that Siebel Open UI uses with cascading style sheets.

Table 33. Cascading Style Sheet Classes in Siebel Open UI

| CSS Class Name | Associated With |
|-------------------------|---|
| appletButton | A form applet button. |
| applet-ButtonDis | A form applet disabled button. |
| applet-form-calculator | A form applet input field that Siebel Open UI configures as a calculator. |
| applet-form-combo | A form applet input field that Siebel Open UI configures as a combo box. |
| applet-form-currency | A form applet input field that Siebel Open UI configures as a currency calculator popup applet. |
| applet-form-date | A form applet input field that Siebel Open UI configures as a date field. |
| applet-form-datetime | A form applet input field that Siebel Open UI configures as a date time field. |
| applet-form-effdat | A form applet input field that Siebel Open UI configures as the effective date. |
| applet-form-mvg | A form applet input field that Siebel Open UI configures for a multi-value group (MVG) applet. |
| applet-form-pick | A form applet input field that Siebel Open UI configures for a pick applet. |
| applet--list-calculator | A list applet input field that Siebel Open UI configures as a calculator. |

Table 33. Cascading Style Sheet Classes in Siebel Open UI

| CSS Class Name | Associated With |
|------------------------|---|
| applet--list-combo | A list applet input field that Siebel Open UI configures as a combo box. |
| applet--list-date | A list applet input field that Siebel Open UI configures as a date field. |
| applet--list-datetime | A list applet input field that Siebel Open UI configures as a date time field. |
| applet--list-efdat | A list applet input field that Siebel Open UI configures as the effective date. |
| applet--list-mvg | A list applet input field that Siebel Open UI configures for an MVG applet. |
| applet--list-pick | A list applet input field that Siebel Open UI configures for a pick applet. |
| drilldown | A list applet field that Siebel Open UI configures as a drill down object. |
| dynatree-root-tag | The root element that Siebel Open UI displays in the tree structure for views. |
| minWidthContainer | The dimensions of a dialog box. |
| screen-tab-deselected | The tab that the user has not chosen in the screen bar. |
| screen-tab-selected | The selected tab that Siebel Open UI displays in the screen bar. |
| searchButton | The button that Siebel Open UI displays with the search control. |
| searchContainer | The container that Siebel Open UI uses for the search field. |
| SearchCtrl | The input field that Siebel Open UI displays for the search control. |
| searchField | The combo box control that Siebel Open UI uses for the search control. |
| searchInput | The input field that Siebel Open UI uses with the search control. |
| sidebarNavButton | The navigation button that Siebel Open UI displays for the sidebar. |
| siebel-nextpage | The next page that Siebel Open UI displays in a tree. |
| siebel-prevpage | The previous page that Siebel Open UI displays in a tree. |
| siebel-selectednode | The selected node that Siebel Open UI displays in a tree. |
| siebui-active-navtab | The Active Navigable tab in the first level and third level of the screen. |
| siebui-applet-active | An active dialog box or applet. |
| siebui-applet-maximize | The applet that Siebel Open UI displays as maximized. |

Table 33. Cascading Style Sheet Classes in Siebel Open UI

| CSS Class Name | Associated With |
|------------------------------|--|
| siebui-appletmenu | An applet menu. |
| siebui-appletmenu-item | A menu item of an applet menu. |
| siebui-appletmenu-separator | An item separator of an applet menu. |
| siebui-applet-minimize | the applet that Siebel Open UI displays as minimized. |
| siebui-applet-nonactive | A nonactive dialog box or applet. |
| siebui-appmenu | An applet menu. |
| siebui-btn-icon-d | A disabled button icon. |
| siebui-btn-icon-e | an enabled button icon. |
| siebui-busy | the mouse cursor. |
| siebui-call-from-ui-disabled | A numerical field on the toolbar when Siebel Open UI does not allow a call. |
| siebui-call-from-ui-enabled | A numerical field on the toolbar when Siebel Open UI does allows a call. |
| siebui-ctrl-drilldown | A form applet field that Siebel Open UI configures as a drill down object. |
| siebui-ctrl-label | A form applet field that Siebel Open UI configures as a label. |
| siebui-ctrl-link | A form applet field that Siebel Open UI configures as a link. |
| siebui-ctrl-mail | A form applet field that Siebel Open UI configures as a mail field. |
| siebui-ctrl-url | A form applet field that Siebel Open UI configures as a URL. |
| siebui-empty-tabs | A view that contains no child tabs. |
| siebui-file-input | A form applet input field that Siebel Open UI uses with a file type. |
| siebui-file-label | A button input field. |
| siebui-input-popup | an input field in a form applet or list applet that Siebel Open UI can call as dialog box. |
| siebui-list-ctrl | A list applet that Siebel Open UI configures as a control. |
| siebui-list-textareactrl | A list applet input field that Siebel Open UI configures as a text area. |
| siebui-mask-content | The styling class that Siebel Open UI uses for the message content when it loads a view or applet. |
| siebui-mask-image | The styling that Siebel Open UI uses to mask an image when it loads an indicator. |
| siebui-mask-overlay | The mask overlay. |

Table 33. Cascading Style Sheet Classes in Siebel Open UI

| CSS Class Name | Associated With |
|------------------------|--|
| siebui-mask-splash | The loading indicator image that Siebel Open UI uses when it loads a page. |
| siebui-menu | An application menu. |
| siebui-menu-item | The menu item of the application menu. |
| siebui-menu-separator | the list separator of the application menu. |
| siebui-nav-links | The jump tab dropdown list. |
| siebui-nav-screenlist | The jump tab that Siebel Open UI displays in a screen. |
| siebui-nav-tabs | Tabs at all levels. |
| siebui-nav-viewlist | The jump tab that Siebel Open UI displays in a view. |
| siebui-popup-currency | The currency calculator dialog box. |
| siebui-popup-input-row | The input fields in the form applet that Siebel Open UI uses with the currency calculator. |
| siebui-popup-label | The labels in the form applet that Siebel Open UI uses with the currency calculator. |
| siebui-row-counter | The row counter that Siebel Open UI displays in a form applet or list applet. |
| ui-state-disabled | A disabled applet menu item. |
| view-tab-deselected | The tab that the user has not chosen in the view bar. |
| view-tab-selected | The tab that the user has chosen in the view bar. |

Other Classes

This topic describes methods that reside in a class that this appendix does not describe elsewhere.

Define Method

The Define method identifies the modules that Siebel Open UI uses to determine the location of the presentation model file or physical renderer file that Siebel Open UI must download to the client. It uses the following syntax:

```
define (module_name , list_of_dependencies, function);
```

where:

- *module_name* is a string that specifies the name of a module.

- *list_of_dependencies* is an array that lists all the modules that *module_name* depends on to run correctly. If no dependencies exist, then this list is not required. For more information, see [“Specifying Dependencies Between Presentation Models or Physical Renderers and Other Files”](#) on page 126.
- *function* identifies a function that must return an object that identifies a function name.

Siebel Open UI recommends that you use the following syntax when you use the define method:

```
if(typeof(" Siebel AppFacade. module_name") === undefined){
  Siebel JS.Namespace("Siebel AppFacade. module_name");
  define(" siebel /custom/module_name", [], function(){
    Siebel AppFacade. module_name = (function(){
      var consts = Siebel JS.Dependency("Siebel App. Constants");
      function module_name() {
        Siebel AppFacade. module_name. superclass.constructor.apply(this,
arguments);
      };
      Siebel JS.Extend(module_name, Siebel AppFacade.arguments_2);
      return module_name;
    })();
    return Siebel AppFacade. module_name;
  });
}
```

where:

- Siebel AppFacade is the name space.
- *module_name* identifies the file name of the presentation model or the physical renderer without the file name extension. For example:

RecycleBINModel

- function defines the class constructor.

You use the Define method when you set up a presentation model or a physical renderer. For an example usage of this method when setting up:

- A presentation model, see [Figure 17 on page 62](#).
- A physical renderer, see [Figure 24 on page 85](#).

For information about how to add manifest files and manifest expressions that reference the *module_name*, see [“Configuring Manifests”](#) on page 128.

Siebel Open UI uses this Define method to make sure that your code meets the requirements that Asynchronous Module Definition (AMD) specifies. For reference information about:

- AMD, see the topic about Asynchronous Module Definition at the following address at the github.com website:

<https://github.com/amdjs/amdjs-api/wiki/AMD>

- AMD, see the topic about AMD at the following address at the requirejs.org website:

<http://requirejs.org/docs/whyamd.html>

- Writing JavaScript with AMD, see the topic about writing modular JavaScript with AMD at the following address at the addyosmani.com website:

<http://addyosmani.com/writing-modular-js/>

Methods for Pop-Up Objects, Google Maps, and Property Sets

This topic describes the methods that Siebel Open UI uses with pop-up objects, Google maps, and property sets. It includes the following information:

- [Pop-Up Presentation Models and Physical Renderers on page 508](#)
- [Method That Integrates Google Maps on page 514](#)
- [Methods That Manipulate Property Sets on page 518](#)

Pop-Up Presentation Models and Physical Renderers

The PopupPModel presentation model specifies how to model pop-up objects. It uses the following syntax:

```
Si ebel App. PopupPModel
```

The PopupRenderer physical renderer specifies how to render pop-up objects. It uses the following syntax:

```
Si ebel AppFacade. PopupRenderer
```

If the status of a reply from the Siebel Server is NewPopup, then Siebel Open UI starts processing this new pop-up object in the client. Siebel Open UI supports modal and nonmodal pop-up objects.

The Popup method specifies how to render pop-up objects. Siebel Open UI typically renders a pop-up object as a dialog box.

Modal Pop-Up Objects

A *modal pop-up object* is a type of pop-up object where the metadata for this object contains all of the following qualities:

- The URL property specifies a Siebel URL.
- The SWE_FULL_POPUP_WINDOW_STR property is false.
- The SWE_FREE_POPUP_STR property is false.

Siebel Open UI can create a modal pop-up in one of the following ways:

- **On the Siebel Server.** URL driven. A multivalue group or pick applet are each an example of a modal pop-up object that Siebel Open UI creates on the Siebel Server. Siebel Open UI sets the value of the URL property to the following HTML attribute of the popup div element:

src

Siebel Open UI does the following work to create a modal pop-up on the server:

- a Calls the loadcontent method to get, and then load the layout from Siebel Server.
- b Initializes and renders the pop-up applet.
- **On the client.** Content driven. The Currency pop-up object is an example of a modal pop-up object that Siebel Open UI creates on the client. Siebel Open UI does the following work to create a modal pop-up on client:
 - c Gets the layout and data for the pop-up object.
 - d Loads the pop-up object into the pop-up dialog box when the user opens this dialog box.

Nonmodal Pop-Up Object

A *nonmodal pop-up object* is a type of pop-up object where the metadata for this object contains any of the following qualities:

- The URL property does not specify a Siebel URL.
- The SWE_FULL_POPUP_WINDOW_STR property is true.
- The SWE_FREE_POPUP_STR property is true.

Siebel Open UI uses a nonmodal pop-up object to open an external URL that it stores as data in a Siebel applet.

Properties of the Pop-Up Presentation Model

Table 34 describes the properties of the PopupPM presentation model. The state, url, and content properties render and maintain the state of the pop-up object. It is recommended that you not set the content and the url properties for the same pop-up object.

Table 34. Properties of the Pop-Up Presentation Model

| Property | Description |
|------------------|---|
| canProcessLayout | Not applicable. |
| closeByXDisabled | Controls the X control of the pop-up object. You can set this property to one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI disables the X control. ■ false. Siebel Open UI enables the X control. |
| content | Contains the HTML source code for the pop-up object. Setting this property configures Siebel Open UI to load the HTML source code into the target, and then to call the Initialize method on the pop-up proxy to update the data. |
| currPopups | Maintains an array of currency pop-ups. |

Table 34. Properties of the Pop-Up Presentation Model

| Property | Description |
|----------------------|--|
| height | Specifies the height of the pop-up object, in pixels. |
| isCancelQryPopupOpen | Includes one of the following return values: <ul style="list-style-type: none"> ■ true. A cancel query object is open. ■ false. No cancel query objects are open. |
| isCurrencyOpen | Includes one of the following return values: <ul style="list-style-type: none"> ■ true. A currency pop-up object is open. ■ false. No currency pop-up objects are open. |
| isPopupClosedByX | Includes one of the following return values: <ul style="list-style-type: none"> ■ true. The user used the X control to close the pop-up object. ■ false. The user did not use the X control to close the pop-up object. |
| isPopupSI | Sets the interactivity mode of the pop-up object. You can set this property to one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI uses standard-interactivity mode. ■ false. Siebel Open UI uses high-interactivity mode. |
| isPrevPopupVisible | Sets the visibility of the parent pop-up object when Siebel Open UI displays a child pop-up object inside the parent. You can set this property to one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI displays the parent. ■ false. Siebel Open UI hides the parent. |
| noHide | Determines whether or not Siebel Open UI can hide the pop-up object. You can set this property to one of the following values: <ul style="list-style-type: none"> ■ true. Siebel Open UI can hide the object. ■ false. Siebel Open UI cannot hide the object. |
| source | Contains the source that Siebel Open UI uses to open the pop-up object. You can set this property to a URL. Siebel Open UI uses this source property to set the url and content properties of this pop-up object. |
| state | Opens or closes the pop-up object. You can set this property to one of the following values <ul style="list-style-type: none"> ■ open. Siebel Open UI opens an empty dialog box. ■ close. Siebel Open UI closes an open dialog box. |

Table 34. Properties of the Pop-Up Presentation Model

| Property | Description |
|----------|--|
| url | <p>Specifies the URL that Siebel Open UI uses to open the pop-up object according to the following mode that the pop-up object uses:</p> <ul style="list-style-type: none"> ■ Modal. Specifies the source URL that contains the content that Siebel Open UI displays in the pop-up object. ■ Nonmodal. Specifies the URL that Siebel Open UI uses to load content into the target HTML element of the pop-up object. <p>Setting this property configures Siebel Open UI to get the layout for this pop-up from the Siebel Server, render this layout, and then to call the Initialize method on the pop-up proxy to load the data.</p> |
| width | Specifies the width of the pop-up object, in pixels. |

Methods of the Popup Presentation Model

Table 35 describes the methods of the PopupPM presentation model. The parentheses that this table includes after each method name lists the arguments that each method supports. An empty set of parentheses indicates that the method supports no arguments.

Table 35. Methods of the Pop-Up Presentation Model

| Method Name | Description |
|--|--|
| ClearPopup() | Sets the pop-up visibility to false and resets various properties and method values after Siebel Open UI closes the pop-up object. |
| OnLoadPopupContent() | Loads the HTML for the pop-up object, initializes pop-up applets, and then calls the show method on the pop-up proxy. |
| OpenPopup(source, height, width, full, free, bContent) | Opens the pop-up object according to the arguments that the ProcessNewPopup method determines. It uses these arguments to set the properties of the pop-up object. Some of these arguments call other methods in the PopupPR physical renderer that load the content in the pop-up object. |
| ProcessClearPopup(propSet) | Calls the ClearPopup method. |

Table 35. Methods of the Pop-Up Presentation Model

| Method Name | Description |
|---------------------------|---|
| ProcessNewPopup(propset) | <p>Processes the property set that Siebel Open UI sends to this method as an argument, and then determines the following items:</p> <ul style="list-style-type: none"> ■ The mode that the pop-up object uses ■ Various pop-up window features ■ The width and height of the pop-up object, in pixels. <p>Siebel Open UI calls the OpenPopup method to open a modal pop-up object. It does not call OpenPopup to open a nonmodal pop-up object. Instead, it creates a nonmodal pop-up object from this ProcessNewPopup method.</p> |
| SetPopupVisible(bVisible) | <p>Modifies the state property of the pop-up object depending on whether the bVisible argument is true or false.</p> |

Methods of the Popup Physical Renderer

Table 36 describes the methods of the PopupRenderer physical renderer.

Table 36. Methods of the Pop-Up Physical Renderer

| Method Name | Description |
|---------------|---|
| BindEvents | <p>Binds all physical events for the pop-up object. For more information, see “Siebel CRM Events You Can Use to Customize Siebel Open UI” on page 563.</p> |
| EnhanceDialog | <p>Resizes the pop-up object according to the width property of the pop-up object and according to the default width that the client specifies.</p> <p>Siebel Open UI calls the EnhanceDialog method when it calls the OnLoadPopupContent method from the PopupPM presentation model.</p> |
| LoadContent | <p>If Siebel Open UI modifies the content property of the PopupPM presentation model, then this LoadContent method loads the HTML source code that contains the content that the pop-up object displays.</p> |
| LoadURL | <p>If Siebel Open UI modifies the url property of the PopupPM presentation model, then this LoadURL method sets the div element of the src attribute of the pop-up object to the value that the url property specifies.</p> |

Table 36. Methods of the Pop-Up Physical Renderer

| Method Name | Description |
|---------------|--|
| SetTitle | <p>Sets the title for the pop-up object according to the following type of client that Siebel Open UI uses:</p> <ul style="list-style-type: none"> ■ high interactivity. Uses the title that it gets from the applet as the title for the pop-up object. ■ standard interactivity. Uses an empty title for the pop-up object. <p>Siebel Open UI calls the SetTitle method when it calls the OnLoadPopupContent method from the PopupPM presentation model.</p> |
| SetVisibility | <p>Displays or hides the pop-up object according to state property of the PopupPM presentation model. If the state property is:</p> <ul style="list-style-type: none"> ■ open. The SetVisibility method displays the pop-up object. ■ close. The SetVisibility method hides the pop-up object. |
| ShowUI | Displays an empty pop-up object. |

Method That Integrates Google Maps

This topic describes the method that Siebel Open UI uses to integrate with Google Maps. It includes the following topics:

- [GetInlineRoute Method on page 514](#)
- [ShowMapLocations Method on page 516](#)
- [Calling Methods That the Integration with Maps and Location Method Uses on page 517](#)

The Integration with Maps and Location service allows the user to view CRM data on a map and get driving directions and other information. If the user taps the postal code of the contact or account address, then Siebel Open UI displays a Google map that includes the address and step-by-step information that describes how to navigate from the current location to the location that the postal code identifies.

This service can get a list of accounts, contacts, or opportunity addresses from the record set that the list applet contains, and then display these addresses in a map. The list view displays distance information from the current location. The map view includes pins on the map that indicate the current location and location of all objects that fall within a radius from the geographic location where the user is currently situated. If the user clicks a pin, then Siebel Open UI does something depending on the following type of information that the pin represents:

- **Opportunity or account.** Navigates the user to details of the record.
- **Contact.** Allows the user to make a telephone call, send an email, or view contact details.

Siebel Open UI uses the google-ui-map plug-in. It includes the following methods in the JQMMapCtrl class:

- `GetInlineRoute`
- `ShowMapLocations`
- `Integration with Maps and Location`

GetInlineRoute Method

The `GetInlineRoute` method does the following work:

- Dynamically loads the Google map method.
- Gets the current location of the device or browser.
- Draws the route. It uses the current location as the starting point and the account location as the destination.

It includes the `DestValue` argument. This argument identifies the postal code or address of an account, contact, or opportunity.

Siebel Open UI calls the predefined `GetInlineRoute` method from a form applet, but you can customize it to use a list applet. The Integration with Maps and Location service creates a link that includes an image and a bind click event that references the control link that calls the `GetInlineRoute` method. It gets the postal code value from the record that the user chooses in the form applet, and then sends the value when it calls the `GetInlineRoute` method in the `JQMMapCtrl` class. Siebel Open UI must load the Google method before it calls the `GetInlineRoute` method. It includes the URL for the Google method when it loads the `JQMMapCtrl` class.

Flow That the `GetInlineRoute` Method Uses

The `GetInlineRoute` method uses the following flow:

- 1 Makes sure the Web template file includes a map div element.
- 2 Calls the `LoadAPI` method.
- 3 Dynamically loads the Google map method. The Google map method is not downloadable so it dynamically loads the map method when it initializes the `JQMMapCtrl` class.
- 4 Calls the `LoadMap` method.
- 5 Removes all markers, overlays, and services from the map div element.
- 6 Creates a Google map in the div element. It uses the div element that it created in the web template. It uses the `google-ui-map` plug-in to create this element in [Step 4](#).
- 7 It sends the name of the `jqmMapCtrl` div element to the plug-in to draw the map.
- 8 Uses the `getCurrentPosition` method to get the current geocode of the client device. This method is available through the `navigator.geolocation` object. A *geocode* is an object that stores the geographic coordinates of a location expressed as latitude and longitude.
- 9 Displays the GPS geocode of the current position. It does this only if the browser supports GPS (Global Positioning System). If the browser does not support GPS, or if GPS is not available, then Siebel Open UI sets the current location to Oracle headquarters at 500 Oracle Parkway, Redwood Shores, CA 94065. The following browsers support GPS:
 - Internet Explorer version 9.0
 - Firefox version 3.5
 - Chrome version 5.0
 - Safari version 5.0
 - Opera version 10.60
- 10 Calls the `GetAcctDirections` method. It uses the following arguments of the `GetAcctDirections` method during this call:
 - **mapCanvas.** Identifies the div element where Siebel Open UI draws the map.
 - **currentLocation.** Identifies the device GPS location. If the browser does not support GPS or if GPS is not available, then it uses the Oracle headquarters address.
 - **acctDestination.** Identifies the postal code or address from the account, contact or opportunity record.

11 Draws the route from the `currentLocation` to `acctDestination`. For example:



ShowMapLocations Method

The `ShowMapLocations` method loads the Google map method, initializes the geocoder service to get the geocode of the address, and creates a marker for each location that the array contains.

It uses the `AcctArray` method. This method gets the address or postal code of all account, contact, or opportunity addresses from the record set that the list applet displays.

Siebel Open UI can call the `ShowMapLocations` method from a list applet. You can create a button or link control, and then bind a click event with the control so that this event calls the method. The `ShowMapLocations` method uses `jqmListRenderert` to do the following work:

- Loop through the record set that the list applet contains
- Determine the columns that are available
- Add the nonnull value of each address field in the record to create the full address.
- Add the address to the array.

You can bind the ShowMap button control in the web template with the click event in `jqmListRenderer`, and then configure Siebel Open UI to use the account array to call the `ShowMapLocations` method in the `JQMMapCtrl` class.

Flow That the ShowMapLocations Method Uses

The `ShowMapLocations` method uses the following flow:

- 1 Calls the `LoadAPI` method that loads the Google map method. The Google map method is not downloadable, so Siebel Open UI loads it when it initializes the `JQMMapCtrl` class and provides the `LoadMap` as a callback method. For more information, see [“Coding Callback Methods” on page 375](#).
- 2 Calls the `LoadAcctsMap` method, which does the following work:
 - a Gets the current geocode of the client device.
 - b Gets the address of the location so that it can display this address in the Info Window.
 - c Creates the Google map in the div element. It uses the div element that it created in the web template. It used the `google-ui-map` plug-in to create this element.
 - d Starts an instance of the Geocoder service.
 - e Does the following work for each address that the `AcctArray` method includes:
 - Gets the geocode of the address.
 - Sets the marker Position according to the geocode.
 - Calls the `addMarker` method to map all markers that the map div element contains.

Calling Methods That the Integration with Maps and Location Method Uses

You can call methods that the Integration with Maps and Location method uses from a form applet or list applet in the following way:

- 1 Initialize the `JQMMapCtrl` class.
- 2 Configure Siebel Open UI so that it sends a single account address or postal code and then binds it to an event that calls the `GetInlineRoute` method.

Siebel Open UI comes predefined to bind the anchor control for the postal code field to a click event. Siebel Open UI displays the postal code field as an icon next to the control. It uses this configuration only for form applets.

To configure a list applet, you must also do the following work:

- a Prepare the account array that stores the addresses or postal codes and bind it to an event that Siebel Open UI can call from a `ShowMapLocations` method. Siebel Open UI comes predefined to concatenate the values of Street Address, City and State fields, and then set the nonnull values that the array contains. It does this so that it can send the array to the method.
- b Create a link or button that calls the method.

Siebel Open UI uses the Google-ui-map plug-in to render the Google map. This plug-in requires a div id to display the map. This div element can reside in any container. The CCViewDetailMap_Mobile.swt file supports list and map rendering. It contains the following code. It uses the jqmMapCtrl div id to render the Google map:

```
<swe:if condition="Web Engine State Properties, IsMobileApplicationMode">
  <div id="Siebel MapContainer" name="Siebel MapContainer"
    style="display: none;">
    <div id="jqmMapCtrl" name="jqmMapCtrl"></div>
  </div>
</swe:if>
```

Methods That Manipulate Property Sets

This topic describes the methods you can use that manipulate property sets. It includes the following information:

- [Structure of the Property Set on page 518](#)
- [AddChild Method on page 519](#)
- [Clone Method on page 519](#)
- [Copy Method on page 519](#)
- [DeepCopy Method on page 520](#)
- [GetChild Method on page 520](#)
- [GetChildByType Method on page 521](#)
- [InsertChildAt Method on page 521](#)
- [RemoveChild Method on page 521](#)
- [SetProperty Method on page 522](#)

Structure of the Property Set

[Table 37](#) describes the structure of the property set that Siebel Open UI uses in the client.

Table 37. structure of the Property Set

| Property | Description |
|--------------|--|
| childArray | Array of all child property sets that the parent property set contains. |
| childEnum | Counter that contains the number of children enumerated in the property set. |
| propArray | Object that contains the values for all properties that the property set contains. |
| propArrayLen | Length of the propArray property. |

Table 37. structure of the Property Set

| Property | Description |
|----------|----------------------------|
| type | Type of the property set. |
| value | Value of the property set. |

AddChild Method

The AddChild method creates a new child property in the property set. It returns one of the following values:

- **true.** Siebel Open UI created a child property.
- **false.** Siebel Open UI did not create a child property.

It uses the following format:

```
AddChild (child)
```

For example:

```
outputPS.AddChild (inputPS);
```

where:

- `inputPS` is an argument that identifies the input property set that Siebel Open UI adds to the `childArray` of the called on property set object `outputPS`.

Clone Method

The Clone method creates a new property set and does a full copy of the following property set:

```
this
```

It returns a new property set object.

It uses the following format:

```
Clone()
```

For example:

```
outputPS = inputPS.Clone();
```

It includes no arguments.

Copy Method

The Copy method copies the following property set:

```
this
```

It returns one of the following values:

- **true.** Siebel Open UI made a copy of the property set.

- **false.** Siebel Open UI did not make a copy of the property set.

It adds every child and subchild in the childArray of the input property set to the childArray of the following property set:

```
this
```

It uses the following format:

```
Copy(old)
```

For example:

```
outputPS.Copy(inputPS);
```

It uses the following arguments:

- **inputPS.** Identifies the input property set that Siebel Open UI copies.

DeepCopy Method

The DeepCopy method makes a full copy of the inputPS property set, and then parses this copy into the following property set:

```
this
```

It returns one of the following values:

- **true.** Siebel Open UI made a full copy of the inputPS property set, and then parsed it.
- **false.** Siebel Open UI did not make a full copy of the inputPS property set, and then parse it.

It uses the following format:

```
DeepCopy(inputPS)
```

For example:

```
outputPS.DeepCopy (inputPS)
```

It uses the following arguments:

- **inputPS.** An input property set that contains the values that Siebel Open UI copies to the outputPS property set.

GetChild Method

The GetChild method returns a child of the property set that resides at an index location that you specify. It returns a property set object.

It uses the following format:

```
GetChild (index)
```

For example:

```
childPS = inputPS.GetChild (index);
```


It uses the following arguments:

- **index.** Specifies the index of the child that Siebel Open UI gets from the inputPS property set.

GetChildByType Method

The GetChildByType method returns a child of the property set according to the type that you specify. It returns a property set object. It uses the following format:

```
GetChildByType (type)
```

For example:

```
childPS = inputPS.GetChildByType("vi ")
```

It uses the following arguments:

- **type.** Specifies the type of the property set that Siebel Open UI gets from the childArray of the inputPS property set.

InsertChildAt Method

The InsertChildAt method inserts a new property set in the child array at the location that the index specifies. It returns one of the following values:

- **true.** Siebel Open UI inserted a new property set.
- **false.** Siebel Open UI did not insert a new property set.

It uses the following format:

```
InsertChildAt (child, index)
```

For example:

```
outputPS.InsertChildAt(inputPS, 2);
```

It uses the following arguments:

- **inputPS.** Specifies the input property set that Siebel Open UI adds in the childArray of the outputPS property set.
- **index.** Specifies the index where Siebel Open UI adds the inputPS property set to childArray.

RemoveChild Method

The RemoveChild method removes a child from the child array of the property set at the location that the index specifies. It returns one of the following values:

- **true.** Siebel Open UI removed a child from the child array.
- **false.** Siebel Open UI did not remove a child from the child array.

It uses the following format:

```
RemoveChild (index)
```

where:

- **index** specifies the index of the child property set that Siebel Open UI removes from the `childArray` of the `outputPS` property set.

For example:

```
outputPS.RemoveChild(2);
```

RemoveProperty Method

The `RemoveProperty` method removes a property from the `propArray` of the property set. It returns one of the following values:

- **true**. Siebel Open UI removed a property from the `propArray`.
- **false**. Siebel Open UI did not remove a property from the `propArray`.

It uses the following format:

```
RemoveProperty (name)
```

where:

- **name** specifies the name of the property that Siebel Open UI removes from `propArray`.

For example:

```
outputPS.RemoveProperty("prop");
```

SetProperty Method

The `SetProperty` method sets a property of the property set. It returns one of the following values:

- **true**. Siebel Open UI set a property of the property set.
- **false**. Siebel Open UI did not set a property of the property set.

It uses the following format:

```
SetProperty (name, value)
```

For example:

```
inputPS.SetProperty("SelectedItem", val);
```

It uses the following arguments:

- **name**. Specifies the new property name.
- **value**. Specifies the new property value.

B

Reference Information for Siebel Open UI

This appendix describes reference information for Siebel Open UI. It includes the following topics:

- [Life Cycle Flows of User Interface Elements on page 523](#)
- [Notifications That Siebel Open UI Supports on page 541](#)
- [Property Sets That Siebel Open UI Supports on page 562](#)
- [Siebel CRM Events You Can Use to Customize Siebel Open UI on page 563](#)
- [Internationalization Support on page 588](#)
- [Screens and Views That Siebel Mobile Uses on page 590](#)
- [Controls That Siebel Open UI Uses on page 595](#)
- [Browser Script Compatibility on page 598](#)

Life Cycle Flows of User Interface Elements

This topic includes flowcharts that you can use to determine the methods that Siebel Open UI uses during various steps in the life cycle of a user interface element. It includes the following information:

- [Life Cycle Flows That Save Records on page 523](#)
- [Life Cycle Flows That Handle User Navigation on page 525](#)
- [Life Cycle Flows That Send Notifications on page 529](#)
- [Life Cycle Flows That Create New Records in List Applets on page 531](#)
- [Life Cycle Flows That Handle User Actions in List Applets on page 535](#)

Life Cycle Flows That Save Records

This topic describes the life cycle flows that Siebel Open UI uses to save records.

Flow That Saves Records If the User Uses a Shortcut

Figure 44 illustrates the life cycle flow that Siebel Open UI uses to save a record if the user simultaneously presses the CTRL and S keys. The numbers in the diagram indicate the sequence that Siebel Open UI uses during this flow. The A connector connects to the flow described in “Flow That Saves Records If the User Uses the Save Menu” on page 525.

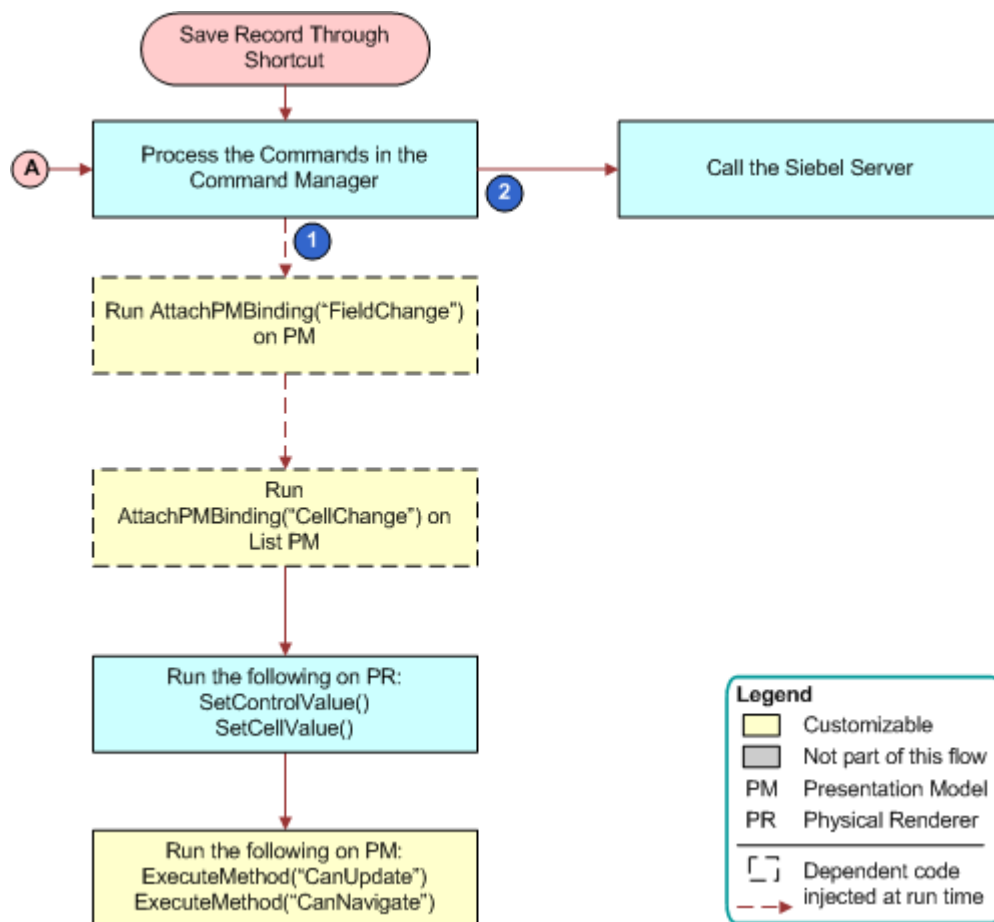


Figure 44. Flow That Saves Records If the User Uses a Shortcut

Flow That Saves Records If the User Uses the Save Menu

Figure 45 illustrates the life cycle flow that Siebel Open UI uses to save a record if the user clicks Menu, and then the Save Record menu item. The A connector connects to the flow described in “Flow That Saves Records If the User Uses a Shortcut” on page 524.

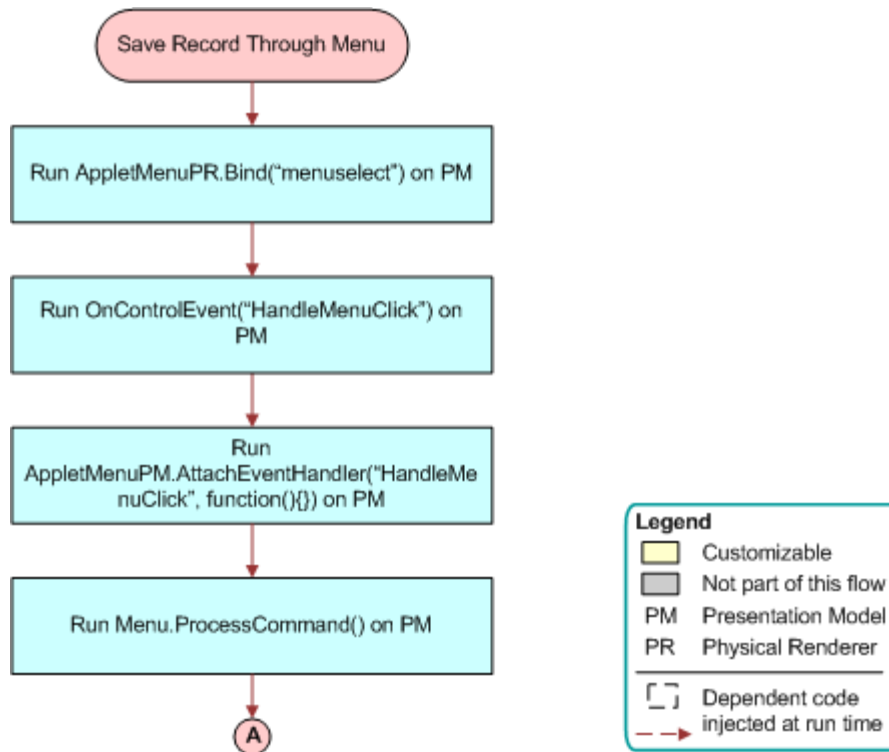


Figure 45. Flow That Saves Records If the User Uses the Save Menu

Life Cycle Flows That Handle User Navigation

This topic describes the life cycle flows that Siebel Open UI uses when the user navigates through various items in the client.

Flow That Siebel Open UI Uses if the User Clicks an Applet in a View

Figure 46 illustrates the life cycle flow that Siebel Open UI uses if the user clicks an applet in a view.

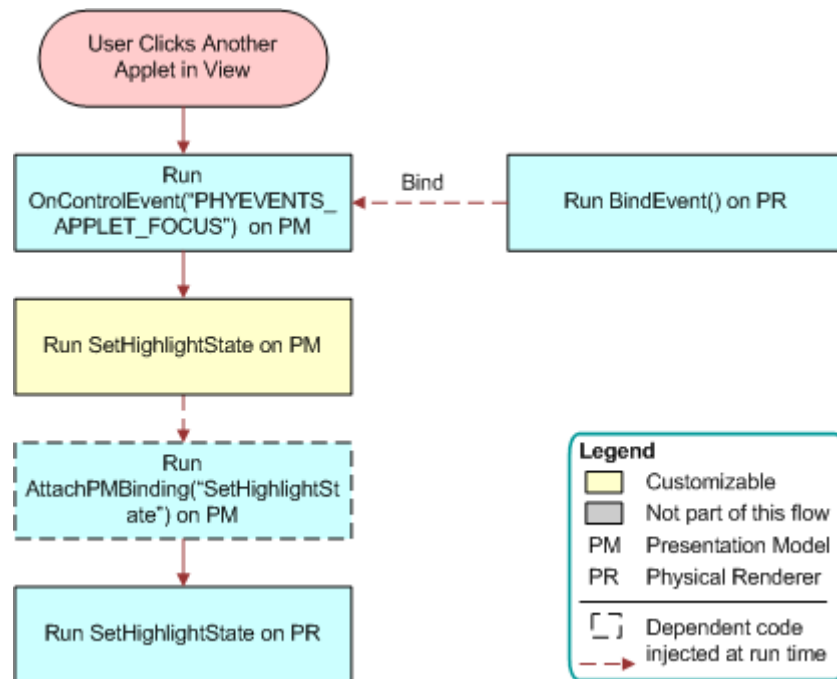


Figure 46. Flow That Siebel Open UI Uses if the User Clicks an Applet in a View

Flow That Siebel Open UI Uses if the User Navigates to a View

Figure 47 illustrates the that Siebel Open UI uses if the user navigates to a view.

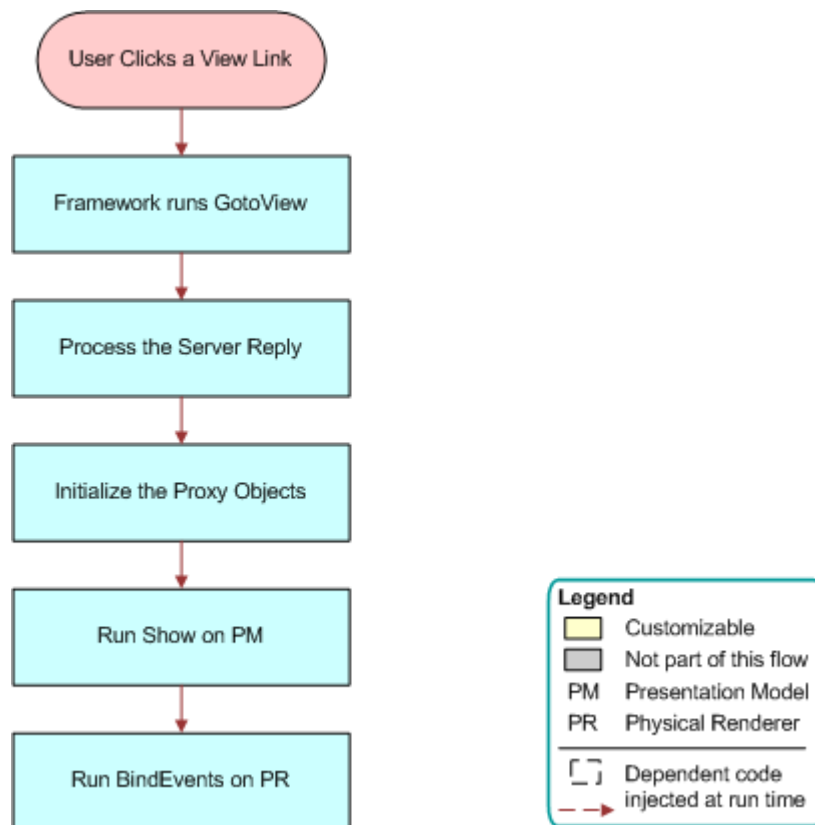


Figure 47. Flow That Siebel Open UI Uses if the User Navigates to a View

Flow That Handles Focus Changes in Form Applets

Figure 48 illustrates the life cycle flow that Siebel Open UI if the focus changes for a field in a form applet. For example, if the user tabs out a field, clicks outside the field, minimizes the window, saves the record, and so on.

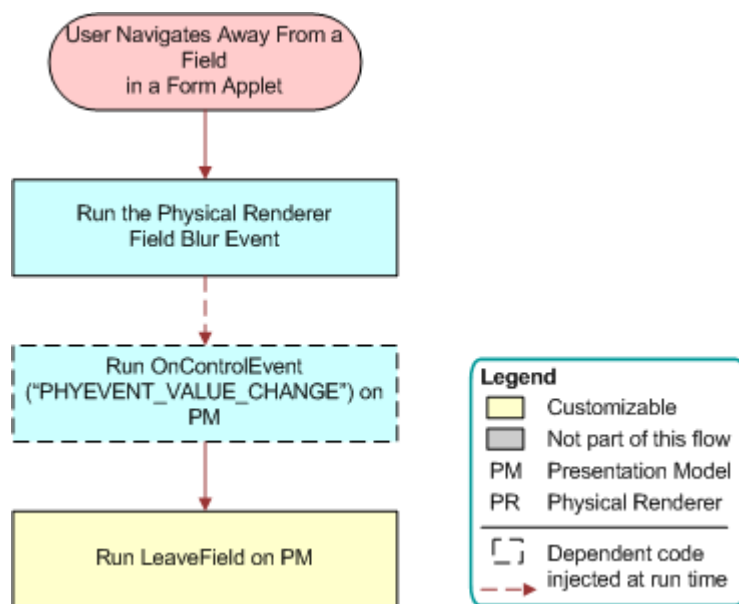


Figure 48. Flow That Handles Focus Changes in Form Applets

Flow That Handles Focus Changes in List Applets

Figure 49 illustrates the life cycle flow that Siebel Open UI if the focus changes for a field in a list applet. For example, if the user tabs out a field, clicks outside the field, minimizes the window, saves the record, and so on.

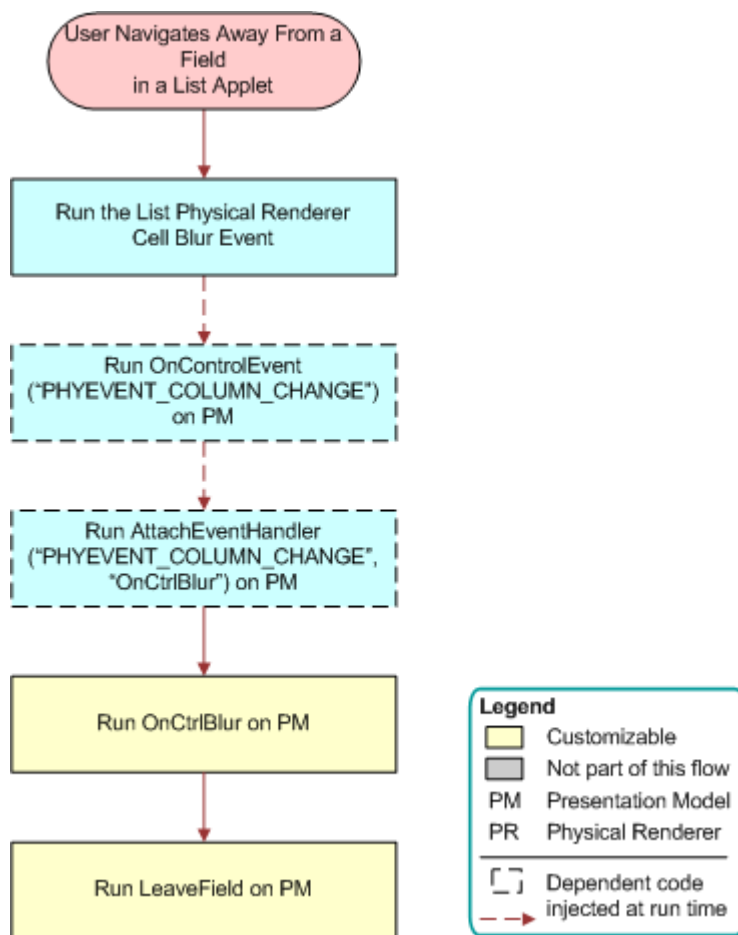


Figure 49. Flow That Handles Focus Changes in List Applets

Life Cycle Flows That Send Notifications

This topic describes the life cycle flows that Siebel Open UI uses to send notifications.

Flow That Notifies the Siebel Server

Figure 50 illustrates the life cycle flow that Siebel Open UI uses to notify the Siebel Server.

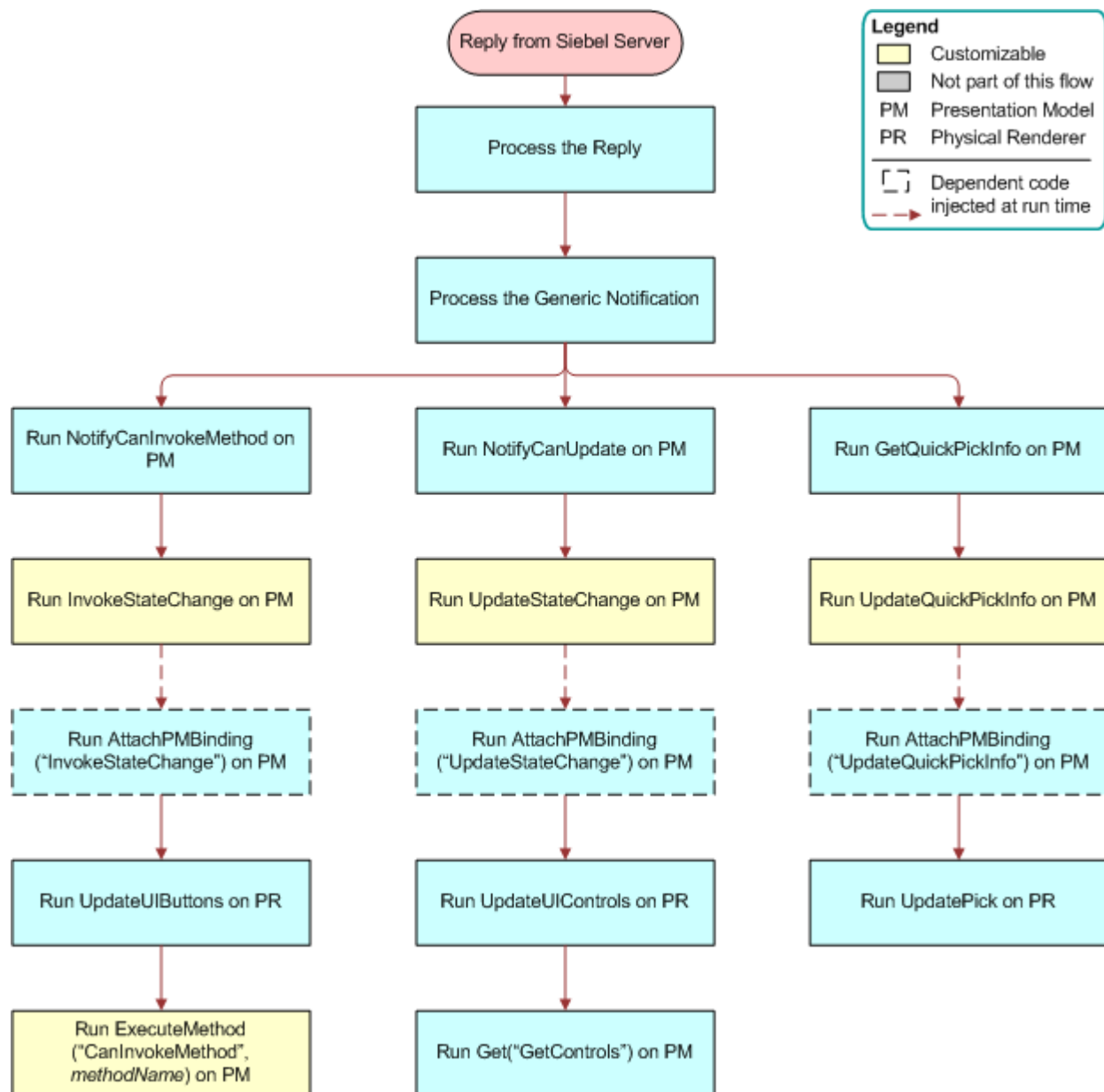


Figure 50. Flow That Notifies the Siebel Server

Flow That Sends a Notification State Change

Figure 51 illustrates the life cycle flow that Siebel Open UI uses to send a notification state change. For more information about the notifications that this flow describes, see [“Notifications That Siebel Open UI Supports”](#) on page 541.

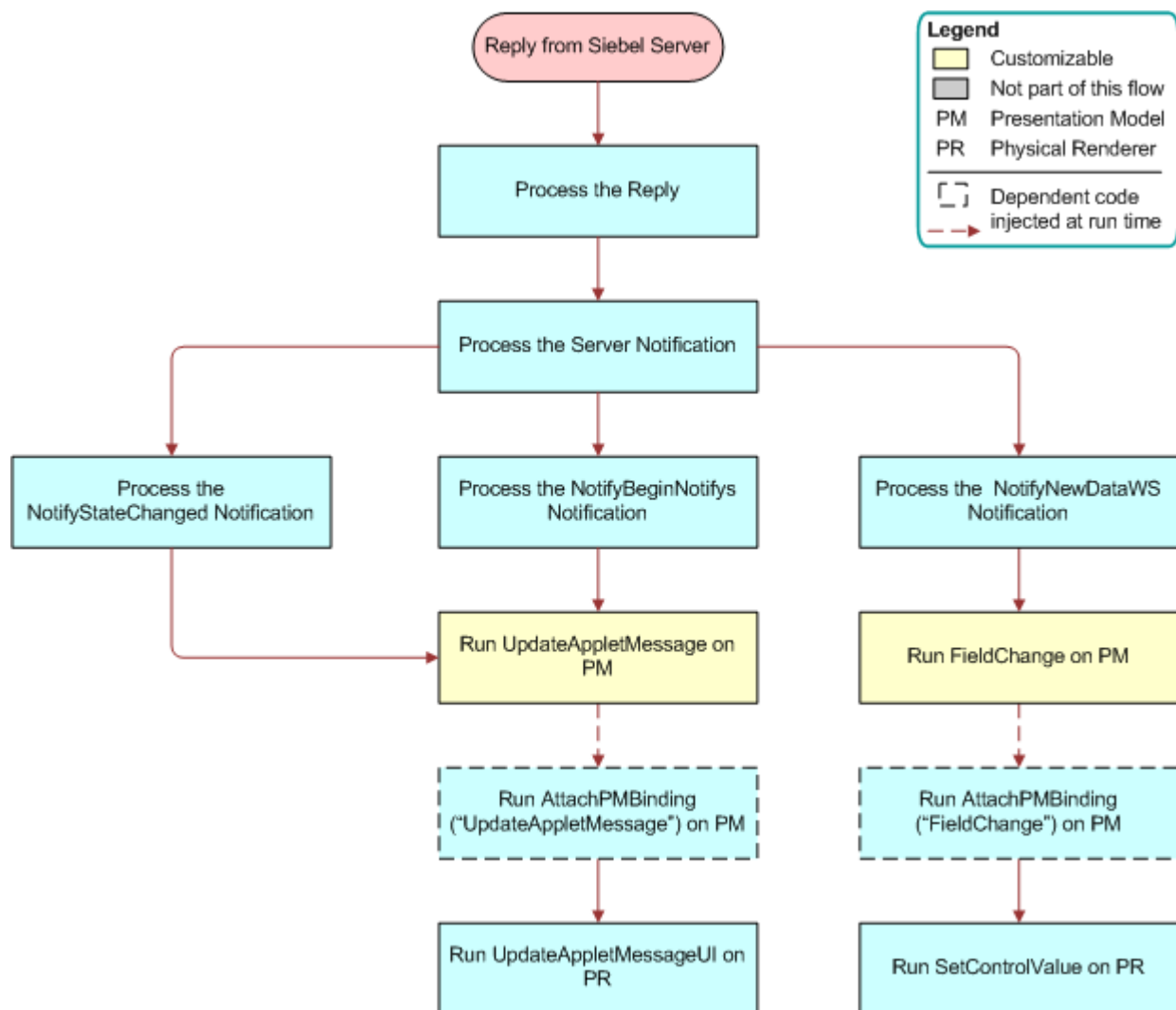


Figure 51. Flow That Sends a Notification State Change

Life Cycle Flows That Create New Records in List Applets

This topic describes the life cycle flows that Siebel Open UI uses to create a new record in a list applet.

Flow That Creates New Records in List Applets, Calling the Siebel Server

Figure 52 illustrates the life cycle flow that Siebel Open UI uses during the call that it makes to the Siebel Server when it creates a new record in a list applet. **Siebel Open UI** typically calls the following methods during this flow: `NewRecord`, `DeleteRecord`, `EditField`, `WriteRecord`, and so on. For more information, see [“DeleteRecord Method” on page 395](#), [“WriteRecord Method” on page 407](#), and [“NewRecord Method” on page 474](#).

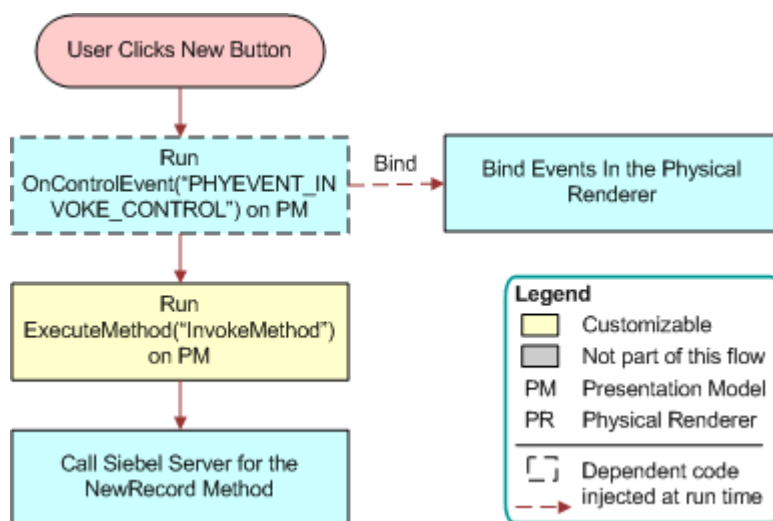


Figure 52. Flow That Creates New Records in List Applets, Calling the Siebel Server

Flow That Creates New Records in List Applets, Processing the Server Reply

Figure 53 illustrates the life cycle flow that Siebel Open UI uses when it processes the reply that it gets from the Siebel Server when it creates a new record in a list applet. This figure illustrates the flow that occurs after Siebel Open UI receives the reply.

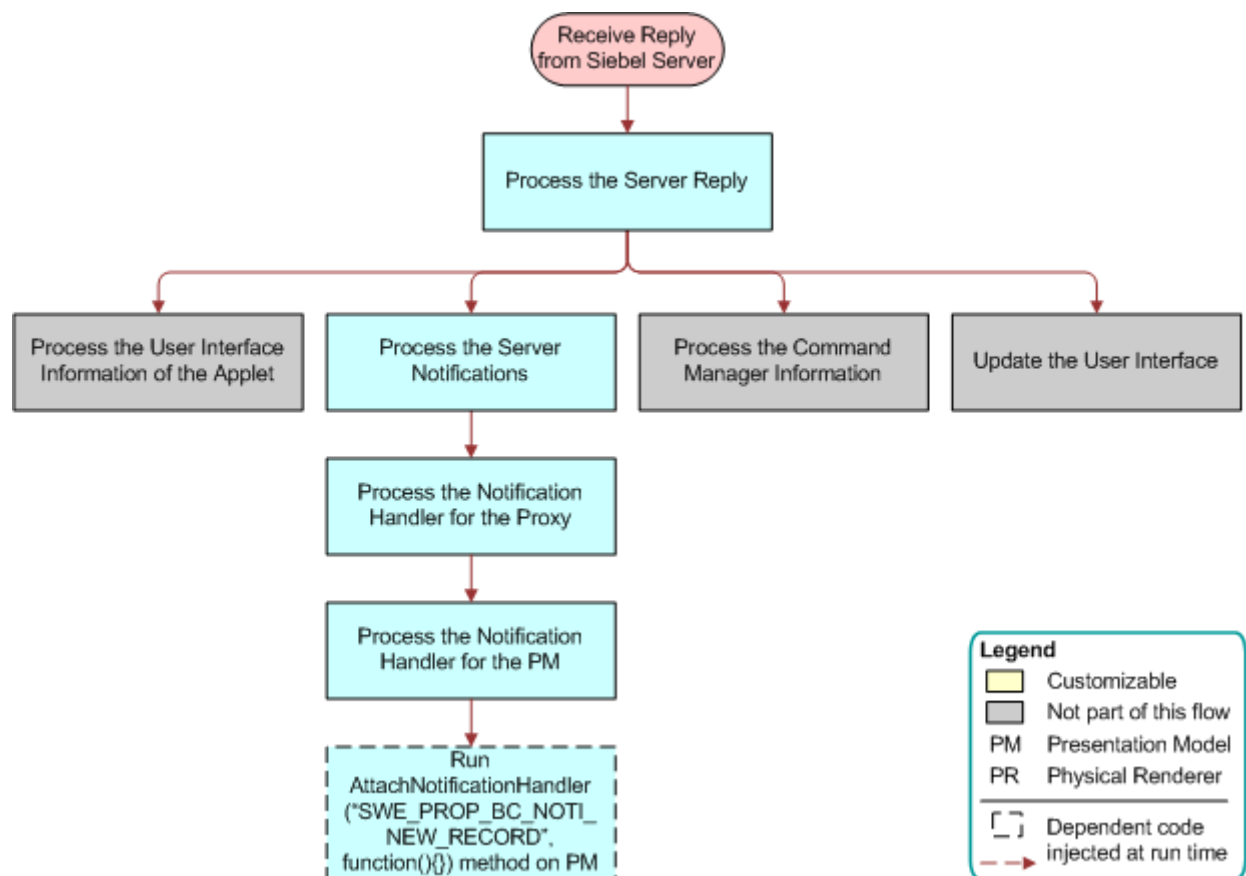


Figure 53. Flow That Creates New Records in List Applets, Processing the Server Reply

Flow That Creates New Records in List Applets, Updating the User Interface

Figure 54 illustrates the life cycle flow that Siebel Open UI uses to update the user interface. The numbers in the diagram indicate the sequence that Siebel Open UI uses during this flow.

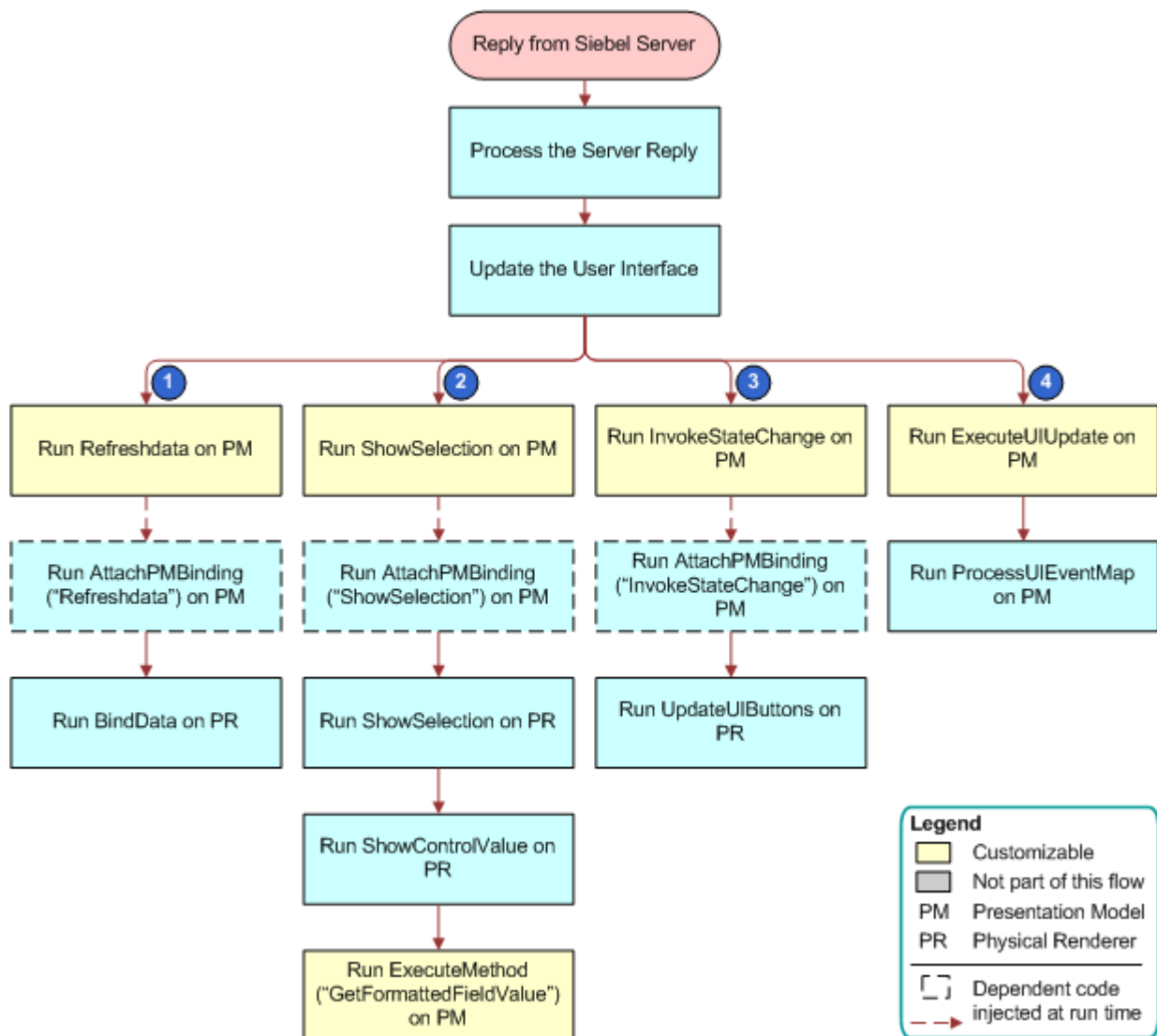


Figure 54. Flow That Creates New Records in List Applets, Updating the User Interface

Flow That Creates New Records in List Applets, Updating the Proxy and Presentation Model

Figure 55 illustrates the life cycle flow that Siebel Open UI uses to update the proxy and presentation model.

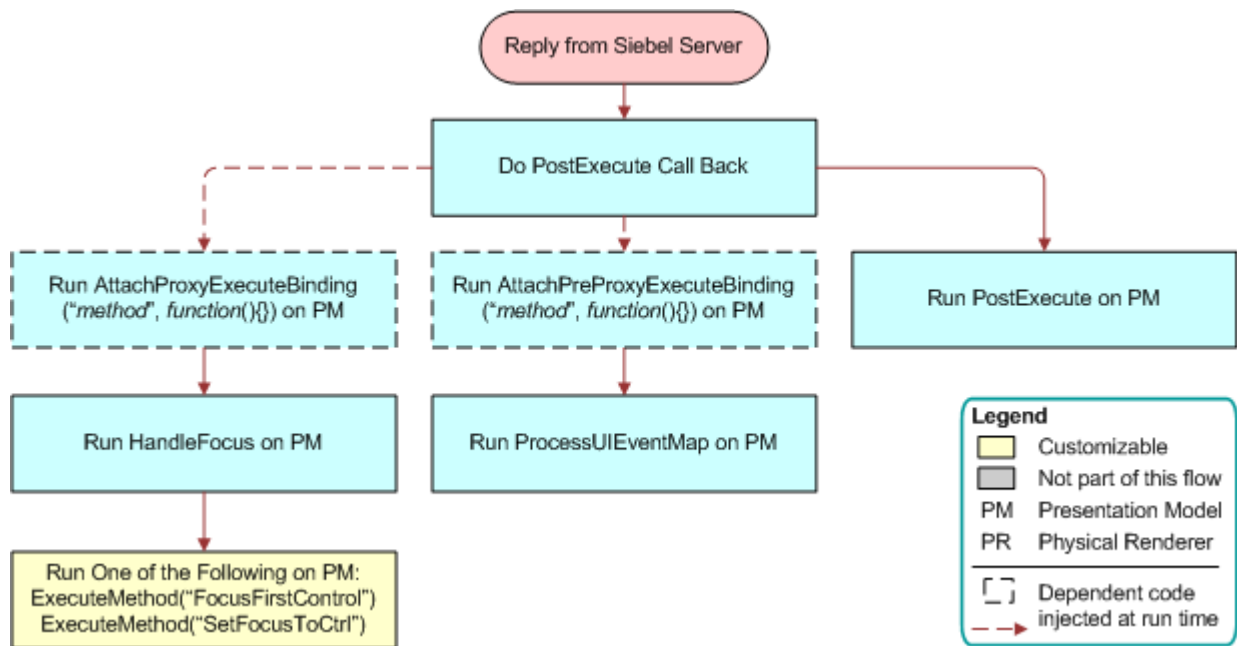


Figure 55. Flow That Creates New Records in List Applets, Updating the Proxy and Presentation Model

Life Cycle Flows That Handle User Actions in List Applets

This topic describes the life cycle flows that Siebel Open UI uses depending on an action that the user does in a list applet.

Flow That Handles Navigation to Another Row in List Applets

Figure 56 illustrates the flow that Siebel Open UI uses if the user navigates to another row in a list applet.

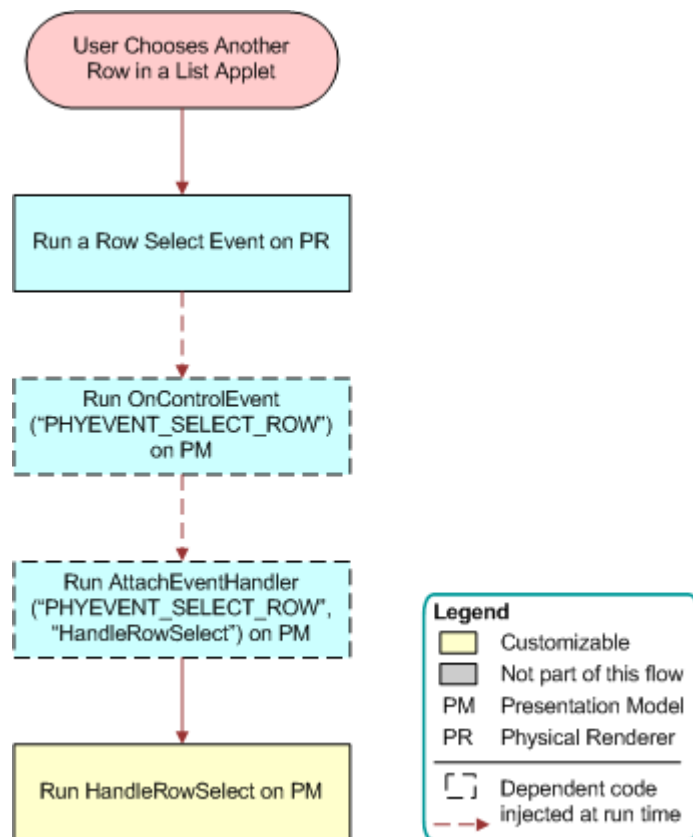


Figure 56. Flow That Handles Navigation to Another Row in List Applets

Flow That Handles the Pagination Button in List Applets

Figure 57 illustrates the flow that Siebel Open UI uses if the user clicks the pagination button in a list applet.

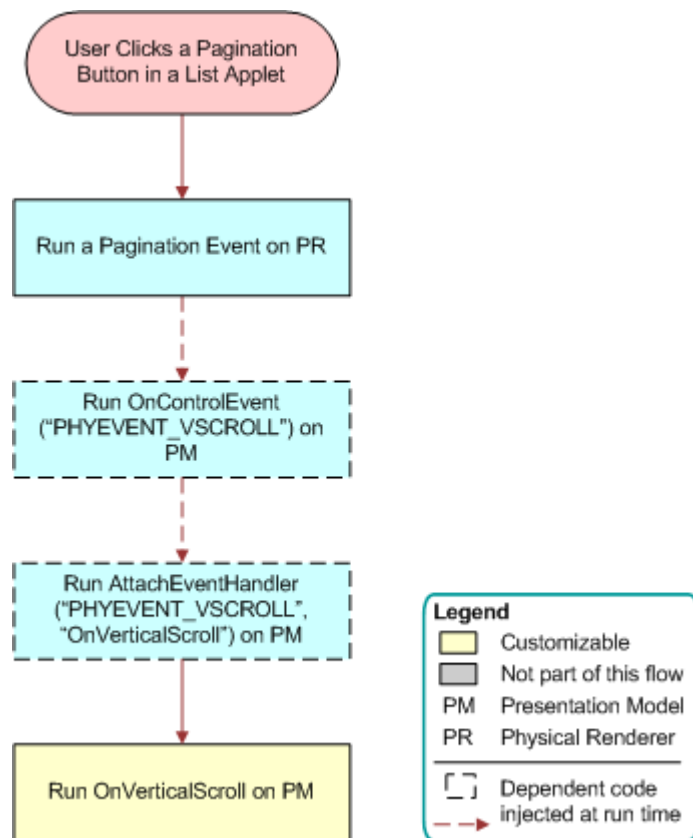


Figure 57. Flow That Handles the Pagination Button in List Applets

Flow That Handles a Column Sort in List Applets

Figure 58 illustrates the flow that Siebel Open UI uses if the user sorts a column in a list applet.

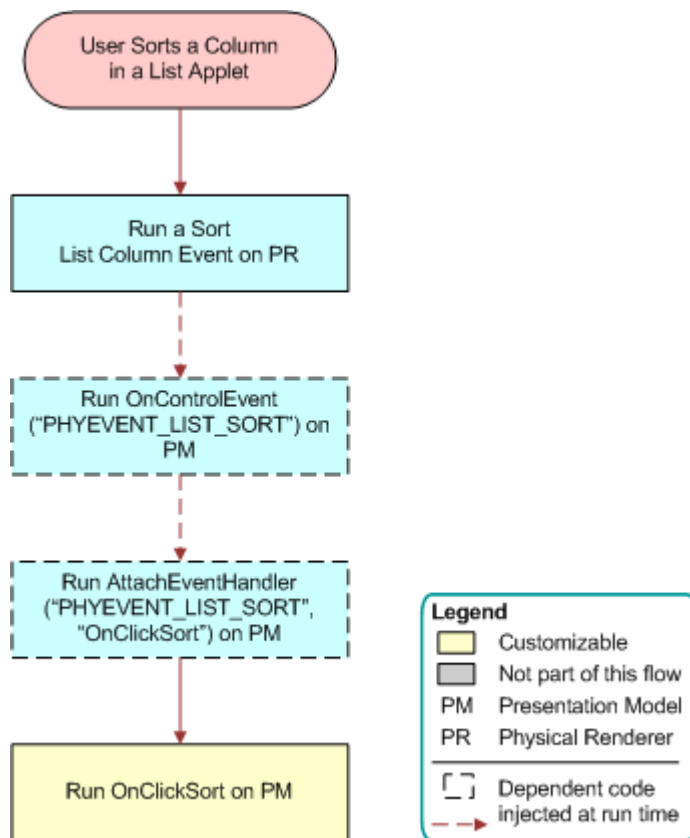


Figure 58. Flow That Handles a Column Sort in List Applets

Flow That Handles a Cell Click in List Applets

Figure 59 illustrates the flow that Siebel Open UI uses if the user clicks a cell in a list applet.

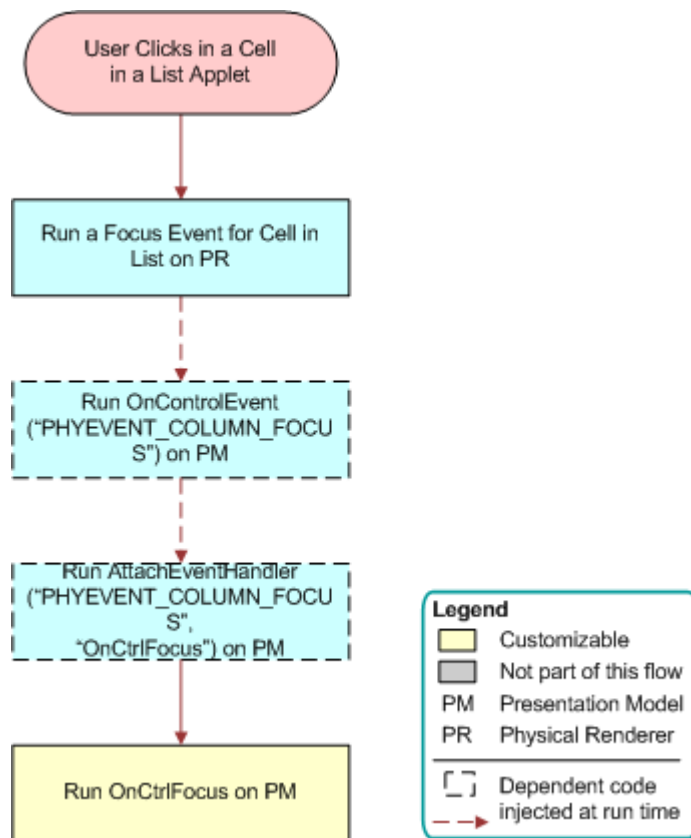


Figure 59. Flow That Handles a Cell Click in List Applets

Flow That Handles a Cell Edit and Blur in List Applets

Figure 60 illustrates the flow that Siebel Open UI uses if the user edits a cell in a list applet, and then navigates away from this cell.

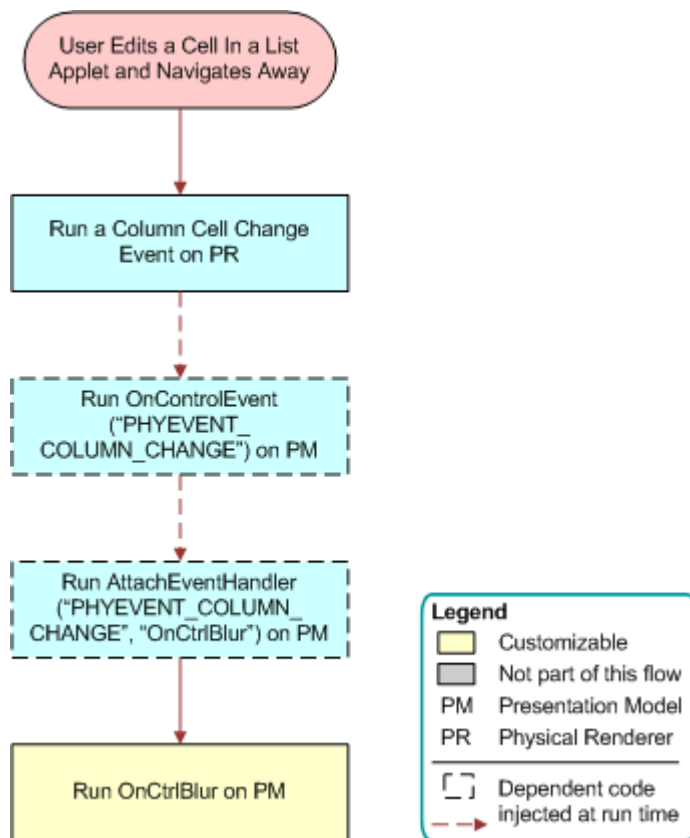


Figure 60. Flow That Handles a Cell Edit and Blur in List Applets

Flow That Handles a Drilldown in List Applets

Figure 61 illustrates the flow that Siebel Open UI uses if the user clicks a drilldown field in a list applet.

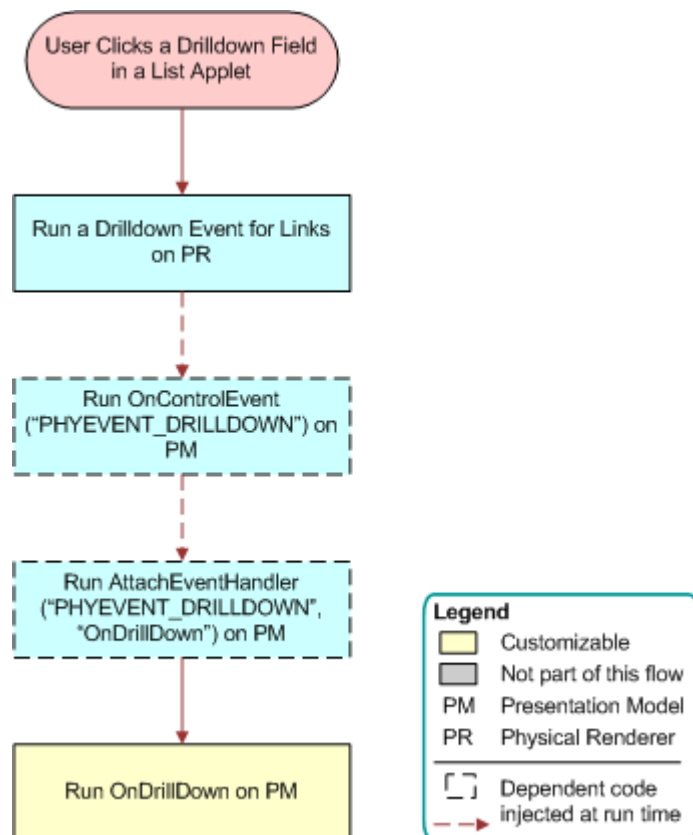


Figure 61. Flow That Handles a Drilldown in List Applets

Notifications That Siebel Open UI Supports

This topic describes notifications that Siebel Open UI supports. It includes the following information:

- [Summary of Notifications That Siebel Open UI Supports on page 542](#)
- [Using Notifications with Operations That Call Methods on page 550](#)
- [NotifyGeneric Notification Type on page 551](#)
- [NotifyStateChanged Notification Type on page 554](#)
- [Example Usage of Notifications on page 557](#)

For more information about configuring Siebel Open UI to use notifications, see [“AttachNotificationHandler Method” on page 428](#).

Summary of Notifications That Siebel Open UI Supports

Table 38 describes the notification types that Siebel Open UI supports. For more information, see [“New Notification User Interfaces” on page 21](#).

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|--------------------|--------------------------------|---|
| NotifyBeginNotifys | SWE_PROP_BC_NOTI_BEGIN | Notifies the client business component that the request that Siebel Open UI sent to the Siebel Server resulted in at least one notification from a business component. |
| NotifyStateChanged | SWE_PROP_BC_NOTI_STATE_CHANGED | <p>Specifies a top-level notification for more than one state change that occurs in the business component level. Siebel Open UI uses the following properties to identify the change and to get the data associated with the change:</p> <ul style="list-style-type: none"> ■ state ■ value <p>Siebel Open UI can provide summary or detailed state information. For more information, see “NotifyStateChanged Notification Type” on page 554.</p> |
| NotifyGeneric | SWE_PROP_BC_NOTI_GENERIC | <p>Identifies the predefined and custom notifications that the Siebel application must send. Siebel Open UI addresses most predefined generic notifications to a particular applet.</p> <p>You can use NotifyGeneric to get the exact type for a generic notification. Siebel Open UI provides actual information of the changes as an encoded argument set.</p> |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|--------------------|---------------------------------|--|
| NotifyNewSelection | SWE_PROP_NOTI_SELECTED | <p>Notifies the client business component that a change occurred in the selection status. Siebel Open UI calls NotifyNewSelection two times for each selection status change:</p> <ul style="list-style-type: none"> ■ One time a value of false for the last row selected ■ One time with a value of true for the new row that Siebel Open UI is selecting <p>You cannot use NotifyNewSelection with a multi-select.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_BC_NOTI_ACTIVE_ROW = index SWE_PROP_NOTI_SELECTED = Boolean</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>index</i> identifies the index of the row that Siebel Open UI is activating or deactivating. ■ <i>Boolean</i> is true or false. |
| NotifyNewActiveRow | SWE_PROP_BC_NOTI_NEW_ACTIVE_ROW | <p>Notifies the client business component that a change occurred on an active row of the corresponding business component on the Siebel Server. Siebel Open UI usually uses NotifyNewSelection with NotifyNewActiveRow.</p> <p>You can use the following syntax:</p> <pre>SWE_PROP_BC_NOTI_ACTIVE_ROW = row</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>row</i> identifies the row that Siebel Open UI is activating or deactivating. |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|--------------------|--------------------------------|--|
| NotifyDeleteRecord | SWE_PROP_BC_NOTI_DELETE_RECORD | <p>Notifies the business component in the client that Siebel Open UI deleted a record from the current set of records on the Siebel Server. Siebel Open UI might use this notification two times for a single record deletion.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_BC_NOTI_ACTIVE_ROW = index bUp = Boolean</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>index</i> identifies the index of a record that resides in the current set of records that Siebel Open UI is deleting. ■ <i>Boolean</i> is one of the following values: <ul style="list-style-type: none"> ■ true. Shift records up after the delete. ■ false. Shift records down after the delete. <p>For an example usage of this notification, see “Customizing the Presentation Model to Handle Notifications” on page 75.</p> |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|-----------------------|---------------------------------|---|
| NotifyDeleteRecordSet | SWE_PROP_BC_NOTI_DELETE_WORKSET | <p>Notifies the business component in the client that Siebel Open UI is deleting a record from the current set of records in the client. Does not correspond to a method invoke. Siebel Open UI sends a separate notification for each record that it deletes.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>i ndex: <i>i ndex</i> NumRows/nr: <i>number</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>index</i> identifies the start index of the record that Siebel Open UI is deleting. ■ <i>number</i> identifies the number of rows that Siebel Open UI must delete. <p>For more information, see “Using Notifications with Operations That Call Methods” on page 550.</p> |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|---------------------|---------------------------------|---|
| NotifyInsertWorkSet | SWE_PROP_BC_NOTI_INSERT_WORKSET | <p>Notifies the business component in the client that Siebel Open UI is inserting a new record in the current set of records in the client.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>index: <i>index_value</i> SWE_FIELD_VALUE_STR: <i>child</i> SWE_PROP_VALUE_ARRAY: <i>array</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>index</i> identifies the index of the record that Siebel Open UI is inserting. ■ <i>child</i> identifies the child property set that contains the record data. ■ <i>array</i> is an array that contains the field values of the record that Siebel Open UI is inserting. This array must use the same sequence that the business component uses when it lists these field values. <p>For more information, see “Using Notifications with Operations That Call Methods” on page 550.</p> |
| NotifyNewData | SWE_PROP_BC_NOTI_NEW_DATA | <p>Notifies the business component in the client that Siebel Open UI is modifying the current set of records. Siebel Open UI sends this notification only if it modifies a record. It does not send this notification if it only modifies a field value.</p> |
| NotifyNewPrimary | SWE_PROP_BC_NOTI_NEW_PRIMARY | <p>Sets the primary record in a multi-value group. The RepopulateField notification calls NotifyNewPrimary.</p> |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|----------------------|-------------------------------------|---|
| NotifyNewRecord | SWE_PROP_BC_NOTI_NEW_RECORD | <p>Notifies the client business component that Siebel Open UI is creating a new record in the current set of records on the Siebel Server. You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_BC_NOTI_ACTIVE_ROW = <i>index</i> insertBefore = <i>Boolean</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>row</i> identifies the index of the record that Siebel Open UI is creating. ■ <i>Boolean</i> is one of the following values: <ul style="list-style-type: none"> ■ true. Place the new record before the previous active row. ■ false. Place the new record after the previous active row. <p>For a similar usage of this notification, see “Customizing the Presentation Model to Handle Notifications” on page 75.</p> |
| NotifyNewRecordData | SWE_PROP_BC_NOTI_NEW_RECORD_DATA | Sets the do populate flag. |
| NotifyNewDataWorkSet | SWE_PROP_BC_NOTI_NEW_RECORD_DATA_WS | Updates a record in the current set of records. |
| NotifyNewFieldData | SWE_PROP_BC_NOTI_NEW_FIELD_DATA | <p>Notifies the client business component that Siebel Open UI modified a field value on the Siebel Server, and that Siebel Open UI communicated this modification to the client through the NotifyNewDataWorkset notification.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_NOTI_FIELD = <i>field</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>field</i> identifies the name of the field that Siebel Open UI is modifying. |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|-----------------------|-------------------------------------|---|
| NotifyNewDataWorkset | SWE_PROP_BC_NOTI_NEW_DATA_WS | <p>Notifies the client business component of a field value that Siebel Open UI modified for a field that resides on the Siebel Server.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_NOTI_FIELD = field SWE_PROP_FIELD_VALUES = child</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>field</i> identifies the name of the field that Siebel Open UI is modifying. ■ <i>child</i> identifies the name of the child property set that contains the modification details. <p>You can use the following syntax in the child property set:</p> <pre>SWE_PROP_FIELD_ARRAY: string1 SWE_PROP_VALUE_ARRAY: string2</pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>string1</i> is an encoded string that identifies the field index. ■ <i>string2</i> is an encoded string that identifies the field value. |
| NotifyNewFieldList | SWE_PROP_BC_NOTI_NEW_FIELD_LIST | Refreshes the entire view internally. |
| NotifyNewRecordDataWS | SWE_PROP_BC_NOTI_NEW_RECORD_DATA_WS | Updates the values in the record set. Siebel Open UI updates the dirty flag during previous notifications. |
| NotifyChangeSelection | SWE_PROP_BC_NOTI_CHANGE_SELECTION | Sets the update conditionals flag and the row counter. |
| NotifyEndNotifys | SWE_PROP_BC_NOTI_END | Notifies the client business component that Siebel Open UI is ending the notification, and that no more server notifications exist for the current transaction. |
| NotifyBeginQuery | SWE_PROP_BC_NOTI_BEGIN_QUERY | Notifies the client business component that Siebel Open UI started a query on the business component on the Siebel Server. |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|-------------------------|--------------------------------------|---|
| NotifyNewQuerySpec | SWE_PROP_BC_NOTI_NEW_QUERYSPEC | Siebel Open UI uses the NotifyNewQuerySpec notification if the user refines a query. If the business component search specification is empty, then NotifyNewQuerySpec clears all field search specifications. |
| NotifyNewFieldQuerySpec | SWE_PROP_BC_NOTI_NEW_FIELD_QUERYSPEC | <p>Notifies the client business component that Siebel Open UI is doing one of the following to query the fields of the current business component on the Siebel Server:</p> <ul style="list-style-type: none"> ■ Using a default query specification ■ Starting or running a query <p>This situation can occur through a predefined or custom configuration, or in reply to a query that the user performs.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>SWE_PROP_NOTI_FIELD = <i>field</i> SWE_PROP_VALUE = <i>search specification</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>field</i> identifies the name of the field that Siebel Open UI is querying. ■ <i>search specification</i> identifies a query specification that is defined on this field. |
| NotifyEndQuery | SWE_PROP_BC_NOTI_END_QUERY | Notifies the client business component that Siebel Open UI is ending a query on the business component on the Siebel Server. This situation can occur if the ExecuteQuery method or the UndoQuery method runs. |

Table 38. Notification Types That Siebel Open UI Supports

| Notification | Type | Description |
|--------------------|--------------------------------|--|
| NotifyExecute | SWE_PROP_BC_NOTI_EXECUTE | <p>Notifies the client business component that Siebel Open UI is running a business component on the Siebel Server.</p> <p>You can use the following syntax in the property set that Siebel Open UI sends:</p> <pre>srt = <i>sort specification</i> s = <i>search specification</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>sort specification</i> identifies the sort specification that Siebel Open UI runs. This sort specification resides on the business component in the SRF. ■ <i>search specification</i> identifies the search specification that Siebel Open UI runs. This search specification resides on the business component in the SRF. |
| NotifyScrollAmount | SWE_PROP_BC_NOTI_SCROLL_AMOUNT | Sets the scroll folder and the amount for a mobile swipe operation. |
| NotifyPageRefresh | SWE_NOTIFY_PAGE_REFRESH | Updates the urltogo with the URL that Siebel Open UI uses to refresh a view. Siebel Open UI gets this URL from a subsequent executeurltogo notification. |

Using Notifications with Operations That Call Methods

It is recommended that you do not use some notifications with an operation that calls a method. For example, if the user paginates to the next page in a set of 10 records, and if you use NotifyInsertWorkSet with the method that calls this pagination, then Siebel Open UI will create 10 separate NotifyInsertWorkSet notifications.

NotifyGeneric Notification Type

Table 39 describes the subtypes of the SWE_PROP_BC_NOTI_GENERIC type that the NotifyGeneric notification type uses. It includes the predefined and custom notifications that a Siebel application must send.

Table 39. NotifyGeneric Notification Type

| Sub Type | Description |
|---------------------|---|
| SWEICanInvokeMethod | Enables the refresh button. |
| SWEICtlDefChanged | Modifies the definition for a control. You can customize Siebel Open UI to dynamically modify the definition that a control uses. For example, modifying a definition from JavaScript text box to a JavaScript combo box. |
| SWEIPrivFlds | Specifies a list of private fields. For example, the Find controls that Siebel Open UI displays in a dialog box. A <i>private field</i> is a type of field that only allows the record owner to view the record. For more information, see <i>Siebel Object Types Reference</i> . |
| SWEICanUpdate | Specifies to display data-driven, read-only behavior. |
| SWEICanNavigate | If a list applet displays zero records, and if the user adds a new record, then the SWEICanNavigate subtype displays the drilldown links. |
| SWEIRowSelection | <p>Sends the set of selected rows that exist in the current set of records to a list applet. You can use the following syntax in the decoded array:</p> <pre>argsArray[0] = <i>applet name</i> argsArray[1-x] = <i>value</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>applet name</i> identifies the name of the applet where Siebel Open UI sends the notification. ■ <i>value</i> is one of the following: <ul style="list-style-type: none"> ■ 1. Indicates selected. ■ 0. Indicates not selected. |

Table 39. NotifyGeneric Notification Type

| Sub Type | Description |
|------------------|--|
| SWEAInvokeMethod | <p>Adds an operation in the life cycle that the InvokeMethod method uses. For example, assume you configure an OK button in an association applet, and that this button closes a dialog box. You can use the SWEAInvokeMethod subtype to configure this button to make a subsequent call to the CreateRecord method if the user clicks OK.</p> <p>You can use the following syntax in the decoded array:</p> <pre>argsArray[0] = <i>applet</i> argsArray[1] = <i>method</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>applet</i> identifies the name of the applet that Siebel Open UI calls during the first operation. ■ <i>method</i> identifies the name of the subsequent method that Siebel Open UI calls. <p>For more information, see “InvokeMethod Method for Presentation Models” on page 443.</p> |
| DeletePopup | Deletes a popup applet. The DeletePopup subtype does not close an applet in the user interface. You can use ClosePopup to close an applet. |
| SetPopupBookmark | Sets the context for a popup bookmark to use the state of the parent applet that resides on the Siebel Server. |

Table 39. NotifyGeneric Notification Type

| Sub Type | Description |
|---------------------|--|
| GetQuickPickInfo | <p>Sends the values of a picklist to an applet. You can use the following syntax in the decoded array:</p> <pre> argsArray[0] = <i>placeholder</i> argsArray[1] = <i>view</i> argsArray[2] = <i>applet</i> argsArray[3] = <i>identifier</i> argsArray[4] = <i>control</i>. argsArray[5-x] = <i>string</i> </pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>placeholder</i> is a placeholder array that you can use. ■ <i>view</i> identifies the name of the view where Siebel Open UI displays the picklist. ■ <i>applet</i> identifies the name of the applet where Siebel Open UI displays the picklist. ■ <i>identifier</i> identifies the HTML identifier of the control that requested the picklist values. ■ <i>control</i> contains one of the following values: <ul style="list-style-type: none"> ■ true. Picklist is associated with the control. ■ false. Picklist is not associated with the control. ■ <i>string</i> contains the values of the picklist. |
| BegRow | <p>Sends the starting row that the Object Manager uses to display the current row in the client. You can use the following syntax in the decoded array:</p> <pre> argsArray[0] = <i>applet name</i> argsArray[1] = <i>value</i> </pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>applet name</i> identifies the name of the applet where Siebel Open UI sends the notification. ■ <i>value</i> contains the value of the beginning row. |
| GetCurrencyCalcInfo | Gets a currency notification from the currency metadata. |
| GetCurrencyCodeInfo | Gets a currency notification from specific currency data. |

Table 39. NotifyGeneric Notification Type

| Sub Type | Description |
|-------------------------|---|
| CloseCurrencyPickApplet | Sends a notification to close a currency applet. |
| ClosePopup | <p>Notifies an applet that Siebel Open UI is closing a popup that is currently open on this applet. You can use the following syntax in the decoded array:</p> <pre>argsArray[0] = <i>applet</i></pre> <p>where:</p> <ul style="list-style-type: none"> ■ <i>applet</i> identifies the name of the applet. |

NotifyStateChanged Notification Type

Table 40 describes the subtypes of the NotifyStateChanged type.

Table 40. NotifyStateChanged Notification Type

| Sub Type | Description |
|-------------------|---|
| activeRow | Identifies the active row of the business component. You can use <i>ar</i> (active row) to abbreviate <i>activeRow</i> . |
| bCanDelete | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can delete a field. ■ 1. The business component cannot delete a field. <p>You can use <i>cd</i> (can delete) as an abbreviation for <i>bCanDelete</i>.</p> |
| bCanInsert | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can insert a field. ■ 1. The business component cannot insert a field. <p>You can use <i>ci</i> (can insert) as an abbreviation for <i>bCanInsert</i>.</p> |
| bCanInsertDynamic | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can insert a dynamic field. ■ 1. The business component cannot insert a dynamic field. <p>You can use <i>cud</i> (can insert dynamic) as an abbreviation for <i>bCanInsertDynamic</i>.</p> |

Table 40. NotifyStateChanged Notification Type

| Sub Type | Description |
|-------------------|---|
| bCanMergeRecords | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. Merge is available in multi select mode. ■ 1. Merge is not available in multi select mode. <p>You can use cm (can merge) as an abbreviation for bCanMergeRecords.</p> |
| bCanQuery | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can query a field. ■ 1. The business component cannot query a field. <p>You can use cq (can query) as an abbreviation for bCanQuery.</p> |
| bCanUpdate | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can update a field. ■ 1. The business component cannot update a field. <p>You can use cu (can update) as an abbreviation for bCanUpdate.</p> |
| bCanUpdateDynamic | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component can update a dynamic field. ■ 1. The business component cannot update a dynamic field. <p>You can use cud (can update dynamic) as an abbreviation for bCanUpdateDynamic.</p> |
| bCommitPending | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. A commit is pending on the business component. ■ 1. A commit is not pending on the business component. <p>You can use cp (commit pending) as an abbreviation for bCommitPending.</p> |
| bDelRecPending | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. A delete is pending on the business component. ■ 1. A delete is not pending on the business component. <p>You can use dp (delete pending) as an abbreviation for bDelRecPending.</p> |

Table 40. NotifyStateChanged Notification Type

| Sub Type | Description |
|-------------------|--|
| bExecuted | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. Siebel Open UI finished processing the business component records. ■ 1. Siebel Open UI did not finish processing the business component records. <p>You can use ex (executed) as an abbreviation for bExecuted.</p> |
| bHasAssocList | <p>Determines whether or not the business component is an association business component. An <i>association business component</i> is a type of business component that includes an intertable.</p> |
| bInMultiSelMode | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component is in multiselect mode. ■ 1. The business component is not in multiselect mode. <p>You can use ms (multiselect) as an abbreviation for bInMultiSelMode.</p> |
| bInQueryState | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component is in a query state. ■ 1. The business component is not in a query state. <p>You can use qs (query state) as an abbreviation for bInQueryState.</p> |
| bInverseSelection | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. An inverse of selection is occurring on the business component. ■ 1. An inverse of selection is not occurring on the business component. <p>You can use is (inverse selection) as an abbreviation for bInverseSelection.</p> |
| bNewRecPending | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. A new record is pending on the business component. ■ 1. A new record is not pending on the business component. <p>You can use np (new record pending) as an abbreviation for bNewRecPending.</p> |
| bNotifyEnabled | <p>Returns a Boolean value that includes one of the following values:</p> <ul style="list-style-type: none"> ■ 0. The business component is not enabled for notifications. ■ 1. The business component is enabled for notifications. <p>You can use n (enabled) as an abbreviation for bNotifyEnabled.</p> |

Table 40. NotifyStateChanged Notification Type

| Sub Type | Description |
|--------------|---|
| NumRows | Returns the number of rows that Siebel Open UI has currently identified. You can use nr (number of rows) as an abbreviation for NumRows. |
| NumRowsKnown | Returns the number of rows that Siebel Open UI has currently identified for a search specification. You can use nrk (number of rows known) as an abbreviation for NumRowsKnown. |
| NumSelected | Returns the number of rows that are currently in multiselect mode. You can use ns (number selected) as an abbreviation for NumSelected. |

Example Usage of Notifications

This topic describes example usage of notifications.

Example of the NotifyBeginNotifys Notification

The following code is an example usage of the NotifyBeginNotifys notification:

```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_DELETE_RECORD"),
function (propSet){
    // Change has occurred at server BC. Do something here:
    this.s.SetProperty ("Refresh_Renderer", true);
});

```

Example of the NotifyNewSelection Notification

The following code is an example usage of the NotifyNewSelection notification:

```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_NOTI_SELECTED"), function
(propSet){
    if (propSet.GetProperty(consts.get("SWE_PROP_NOTI_SELECTED")) === "false")
        this.s.SetProperty ("rowBeingUnselected",
propSet.GetProperty("SWE_PROP_BC_NOTI_ACTIVE_ROW"));
    }
});

```

Example of the NotifyNewFieldData Notification

The following code is an example usage of the NotifyNewFieldData notification:

```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_NEW_DATA_WS"),
function (propSet){
    var field = propSet.GetProperty(consts.get("SWE_PROP_NOTI_FIELD"));
    if (field === "Customer Last Name"){
        // Notify my extension that shows this value in a different way.
    }
});

```

```

        this.SetProperty ("RefreshExtn", true);
    }
}

```

Example of the NotifyNewDataWorkset Notification

The following code is an example usage of the NotifyNewDataWorkset notification:

```

// Trap an incoming change to the field value to do some flagging.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_NEW_DATA_WS"),
function (propSet){
    var field = propSet.GetProperty(consts.get("SWE_PROP_NOTI_FIELD"));
    if (field === "Discount Percentage"){
        var childPS = propSet.GetChildByType (consts.get("SWE_PROP_FIELD_VALUES"));
        var value;
        CCFMiscUtil.StringToArray
        (fieldSet.GetProperty(consts.get("SWE_PROP_VALUE_ARRAY")), value);
        if (parseFloat(value) > 20){
            // Greater than 20% discount? Something fishy!
            this.SetProperty ("flagCustomer", true);
        }
    }
});

```

Example of the NotifyNewData, NotifyInsertWorkSet, and NotifyDeleteRecordSet Notifications

The following code is an example usage of the NotifyNewData, NotifyInsertWorkSet, and NotifyDeleteRecordSet notifications:

```

// First let's check if there's any change to the client workset.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_NEW_DATA"), function
(){
    // Yes indeed.
    this.SetProperty ("primeRenderer", true);
});
// Now let's say our business is with the 4th record. We want to track if this record
is replaced.
// First we see if this 4th record is being deleted.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_DELETE_WORKSET"),
function (propSet){
    if (propSet.GetProperty("index" === 3){// 3 because count starts at 0
        if (propSet.GetProperty("nr") === 1 || propSet.GetProperty("NumRows") === 1){
            if (this.Get("primeRenderer")){
                this.Set("deleted", 4);
            }
        }
    }
});
// Next to the insertion
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_INSERT_WORKSET"),
function (propSet){
    var underReplacement = this.GetProperty ("deleted");

```

```

    if (this.Get("primeRenderer") && propSet.GetProperty("index") ===
this.GetProperty("deleted")){
    // All conditions met. Now we'll get some info from what is being added.
    var childPS = propSet.GetChildByType (consts.get("SWE_FIELD_VALUE_STR"));
    var fieldArray;
    CCFMiscUtil.StringToArray
(childPS.GetProperty(consts.get("SWE_PROP_VALUE_ARRAY")), fieldArray);
    this.SetProperty ("New_Name_Value", fieldArray[2]);
    this.SetProperty ("primeRenderer", false);
}
});

```

Example of the NotifyBeginQuery, NotifyNewFieldQuerySpec, and NotifyEndQuery Notification

The following code is an example usage of the NotifyBeginQuery, NotifyNewFieldQuerySpec, and NotifyEndQuery notifications:

```

// Let's see a simple example involving all the three. First we will take up the
query start.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_BEGIN_QUERY"),
function (){
    // Query begins - The renderer will use this notification to show a bubble box
having a number of choices. This might be driven off of a dropdown in the actual
applet - we already have the choices.
    this.SetProperty ("showBubble", true);
});
// Now we'll attach to Field Spec. If that dropdown has a pre default value, then
we can highlight that choice in our bubble.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_NEW_FIELD_QUERYSPEC"
), function (propSet){
    if (propSet.GetProperty(consts.get("SWE_PROP_NOTI_FIELD") === "Customer Type"){
        var value = propSet.GetProperty(consts.get("SWE_PROP_VALUE"));
        var bubbleValues = this.Get (bubbleValueArray);
        this.SetProperty ("SetBubbleHighlightIndex", bubbleValues.indexOf(value));
    }
});
// Next the obvious. The death of the bubble.
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_END_QUERY"), function
(){
    this.SetProperty ("showBubble", false);
});

```

Example of the NotifyEndNotifys Notification

The following code is an example usage of the NotifyEndNotifys notification:

```

this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_END"), function (){
    // No more notifications. Mark for UI Refresh.
    this.SetProperty ("refreshUI", true);
});

```

Example of the SWEIRowSelection Notification

The following code is an example usage of the SWEIRowSelection notification:

```
this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
    var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
    if (type === "SWEIRowSelection"){
        var argsArray;
        CCFMIScUtil.StringToArray
(propSet.GetProperty(consts.get("SWE_PROP_ARGS_ARRAY"), argsArray);
        if (argsArray[6] === "1"){
            this.SetProperty ("SixthRowSelected", true);
        }
    }
});
```

Example of the BegRow Notification

The following code is an example usage of the BegRow notification:

```
this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
    var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
    if (type === "BegRow"){
        var argsArray;
        CCFMIScUtil.StringToArray
(propSet.GetProperty(consts.get("SWE_PROP_ARGS_ARRAY"), argsArray);
        this.SetProperty ("beginRow", parseInt(argsArray[1]));
    }
});
```

Example of the GetQuickPickInfo Notification

The following code is an example usage of the GetQuickPickInfo notification:

```
this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
    var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
    if (type === "GetQuickPickInfo"){
        var argsArray;
        CCFMIScUtil.StringToArray
(propSet.GetProperty(consts.get("SWE_PROP_ARGS_ARRAY"), argsArray);
        if (argsArray[5] === "MyValue"){
            // Ah so the dropdown contains a value that we dont like..
            // Let us be evil and disable it!
            this.SetProperty ("disablePick", true);
        }
    }
});
```

Example of the ClosePopup Notification

The following code is an example usage of the ClosePopup notification:


```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
    var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
    if (type === "ClosePopup"){
        // The below is just an example. PM's should not alert anything. Leave that to
        the PRs.
        alert ("Make sure you have collected all details. Your next operation will save the
        record!");
    }
});

```

Example of the SWEAInvokeMethod Notification

The following code is an example usage of the SWEAInvokeMethod notification:

```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
    var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
    if (type === "SWEAInvokeMethod"){
        if (argsArray[1] === "NewRecord"){
            // Ah so there's going to be a new record that has happened as a chain.
            // Let's set a property so that we can do an AddMethod on this NewRecord,
            // and extend it to our liking
            this.s.SetProperty ("isChained", true);
        }
    }
});

```

Example of the NotifyStateChanged Notification

The following code is an example usage of the NotifyStateChanged notification:

```

this.s.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_STATE_CHANGED"),
function (propSet){
    var type = propSet.GetProperty("type");
    var value = propSet.GetProperty("value");
    // This is just an example. Switch-case is preferred if multiple types need to
    // have custom logic
    if (type === "cr" || type === "CurRowNum"){
        // Current Row has changed at the server.
        if (value === this.s.Get("MyPreviouslyStoredActiveRow")){
            // Or not! What's going on. Something wrong with my logic?
        }
    }
    if (type === "nr" || type === "NumRows"){
        if (parseInt(value) > 1000){
            // Woah, this user seems to be going through a lot of records!
            // Shouldn't she be using the query function?
            this.s.SetProperty ("AlertUser", true);
        }
    }
}

```

```

    ...
    ...
});

```

Property Sets That Siebel Open UI Supports

Table 41 describes the property sets that Siebel Open UI supports. Siebel Open UI uses the Handle Response property set for navigation controls, such as the predefined query, drop down menus, screen tabs, and view tabs. Siebel Open UI handles the toolbar during setup because it does not dynamically update the property set.

Table 41. Property Set Types That Siebel Open UI Supports

| Property Set | Description |
|-----------------------------|--|
| SWE_PROP_NC_PDQ_INFO | <p>Child property set for the Predefined Query (PDQ). It includes the list of PDQ items that Siebel Open UI displays. It uses the following properties:</p> <ul style="list-style-type: none"> ■ SHOW_EMPTY_STRING. Display an empty PDQ entry. ■ SWE_PROP_NC_CAPTION. Identifies the caption that Siebel Open UI displays for the PDQ. |
| SWE_PROP_NC_VISIBILITY_INFO | <p>Defines the beginning of the visibility property set. It uses the following properties:</p> <ul style="list-style-type: none"> ■ SWE_PROP_NC_ITEM_INFO. Identifies the item start property. ■ SWE_PROP_NC_SCREEN_NAME. Identifies the screen name. ■ SWE_PROP_NC_VIEW_NAME. Identifies the view name. ■ SWE_PROP_NC_CAPTION. Identifies the display value for the caption. ■ SWE_PROP_NC_SCREEN_TAB_ICON. Identifies the icon name to use for the screen tab. <p>You can use the VisDropDownItem property as the predefined bubbled handler to do user interface binding.</p> |
| SWE_PROP_NC_SCREENCTRL_INFO | <p>Defines information for the first level in a screen. It uses all the same properties that the SWE_PROP_NC_VISIBILITY_INFO property set uses except it does not use the SWE_PROP_NC_ITEM_INFO property.</p> <p>You can use the GetTabInfo property as the predefined bubbled handler.</p> |

Table 41. Property Set Types That Siebel Open UI Supports

| Property Set | Description |
|-------------------------------|---|
| SWE_PROP_NC_FLOATING_TAB_INFO | If Siebel Open UI already displays a screen, and if the user clicks an empty tab, then SWE_PROP_NC_FLOATING_TAB_INFO adds a new tab. It uses the GetTabInfo property. |
| SWE_PROP_NC_AGGREGATE_INFO | Describes Amazon link information. It uses the GetTabLinkInfo property. It uses the following properties: <ul style="list-style-type: none"> ■ SWE_PROP_NC_VIEW_NAME. Specifies the view name that Siebel Open UI displays. ■ SWE_PROP_NC_CAPTION. Specifies the caption that Siebel Open UI displays. |
| SWE_PROP_NC_SUBDETAIL_INFO | Specifies information for the fourth level link. It uses the same properties that the SWE_PROP_NC_AGGREGATE_INFO property set uses. |
| SWE_PROP_NC_DETAIL_INFO | Specifies the view tab information that Siebel Open UI displays to start. It uses the GetTabInfo property. It uses the following properties: <ul style="list-style-type: none"> ■ SWE_PROP_NC_VIEW_NAME. View name that Siebel Open UI displays. ■ SWE_PROP_NC_CAPTION. Caption that Siebel Open UI displays. You can use the following properties as predefined bubbled handlers: <ul style="list-style-type: none"> ■ GetDataReloadDirty. Specifies the flag property. ■ GetSelectedTabKey. Selection of tabs. ■ GetSelectedTabLinkKey. Selection of links underneath tabs. ■ GetTabInfo. All tabs. ■ GetTabLinkInfo. All links. |

Siebel CRM Events You Can Use to Customize Siebel Open UI

This topic describes the Siebel CRM events that you can use to customize Siebel Open UI. The jQuery library binds actions to JavaScript events, such as mousedown, mouseover, blur, and so on. It also provides its own events that are part of the jQuery library and not part of Siebel Open UI. Siebel Open UI uses some Siebel CRM events that jQuery does not define, such as the postload event. This topic describes these Siebel CRM events. Note the following:

- All example code that this topic describes must reside in the Init method of your custom presentation model. For more information, see [“Init Method” on page 431](#).

- You can use JavaScript methods to manage Siebel CRM events, such as `BindEvent`, `OnControlEvent`, and so on. For more information, see [“OnControlEvent Method” on page 432](#), [“BindEvents Method” on page 461](#) and [“AttachEventHandler Method” on page 427](#).

Events That You Can Use to Customize Form Applets

Table 42 describes the events that you can use to customize a form applet.

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|---|
| PHYEVENT_APPLET_FOCUS | <p>Siebel Open UI triggers the PHYEVENT_APPLET_FOCUS event when an applet receives focus. You can use it to trigger custom code when Siebel Open UI sets the focus to the presentation model that the applet references as the result of a user action. No objects are available with this event. The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebc consts.get("PHYEVENT_ APPLET_FOCUS"), function () { // My applet received focus. this.SetProperty ("AppletInFocus", true); return (true); });" </pre> |
| PHYEVENT_CONTROL_FOCUS | <p>Siebel Open UI triggers the PHYEVENT_CONTROL_FOCUS event when a control in an applet comes into focus. You can use this event to update the value for this control or to call code according to this value. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object that receives focus. ■ value. The value of the control object that receives focus. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(siebc consts.get("PHYEVENT_CO NTROL_FOCUS"), function (control, value) { if (this.Get("AppletInFocus"){ var controlArray = this.Get("FlaggedControlSet"); if (controlArray.indexOf(control) >= 0){ // This is a flagged control. var valueObject = this.GetProperty ("FlaggedControl sValue"); if (value > maxVal ue){ // Value higher than allowed when recei vi ng focus. Let's flag this. this.SetProperty ("FlagHi gher", true); } } } return (true); }); </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|-----------------------|--|
| PHYEVENT_CONTROL_BLUR | <p>Siebel Open UI triggers the PHYEVENT_CONTROL_BLUR event when a control in an applet goes out of focus. You can use this event to update the value for this control or to call code according to this value. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object that lost focus. ■ value. The value of the control object that lost focus. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebConsts.get("PHYEVENT_ CONTROL_BLUR"), function (control, value) { if (this.Get("AppletInFocus")){ var controlArray = this.Get("FlaggedControlSet"); if (controlArray.indexOf(control) >= 0){ // This is a flagged control. var valueObject = this.GetProperty ("FlaggedControl sValue"); if (valueObject[control.GetName()] !== value){ // Value change. Need to update internal storage, and fire any potential extensions attached to the property. valueObject[control.GetName()] = value; this.SetProperty ("FlaggedControl sValue", valueObject); } } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|-------------------------|--|
| PHYEVENT_INVOKE_CONTROL | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_CONTROL event when it calls a method that is associated with a control. Siebel Open UI makes this call in reply to a user action. You can use this event to configure Siebel Open UI to call a method at the physical layer before it makes this call at a proxy layer. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ method. The method that Siebel Open UI calls. ■ inputPS. The input property set that Siebel Open UI sends to the method that it calls. ■ ai. For more information, see “Values That You Can Set for the AI Argument” on page 486. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebConsts.get("PHYEVENT_ INVOKE_CONTROL"), function (method, inputPS, ai) { if (method === "WriteRecord"){ var valueObject = this.GetProperty ("FlaggedControl sValue"); var min = this.Get("MinRangeForFlagged"); for (var value in valueObject){ if (value < min){ alert ("Invalid Values. Think again!"); return (false); } } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|----------------------|---|
| PHYEVENT_INVOKE_PICK | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_PICK event when it calls a pop-up control for a picklist. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the action that the pop-up control requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the picklist that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebelConsts.get("PHYEVENT_ INVOKE_MVG"), function (control) { if (control === this.GetProperty("AddressMVG")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, can't popup this MVG"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|---------------------|---|
| PHYEVENT_INVOKE_MVG | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_MVG event when it calls a pop-up control for a multivalue group. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the action that the pop-up control requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the multivalue group that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebcConsts.get("PHYEVENT_ INVOKE_MVG"), function (control) { if (control === this.GetProperty("AddressMVG")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, can't popup this MVG"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|---|
| PHYEVENT_INVOKE_DETAIL | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_DETAIL event when it calls a pop-up details control. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the action that the pop-up control requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the pop-up details control that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(siebConsts.get("PHYEVENT_INVOKE_DETAIL"), function (control) { if (control === this.GetProperty("City")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, can't use this Details"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|--|
| PHYEVENT_INVOKE_EFFDAT | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_EFFDAT event when it calls a pop-up effective dating control. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the action that the pop-up control requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the effective dating pop-up control that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebcConsts.get("PHYEVENT_ INVOKE_EFFDAT"), function(control){ if(control=== this.GetProperty("AccountTrail")){ var highValue = this.Get("HighVal"); if(highValue < this.Get ("MinRangeForFlagged")){ alert("Sorry, can't open this Dating Popup"); return(false); } } return(true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|-----------------------|--|
| PHYEVENT_INVOKE_COMBO | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_COMBO event when it calls a combo box or dropdown list. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the open action that the combo box or dropdown list requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the combo box or dropdown list that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebConsts.get("PHYEVENT_ INVOKE_COMBO"), function (control) { if (control === this.GetProperty("Designation")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged"){ alert ("Sorry, can't open this Dropdown"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|--------------------------|---|
| PHYEVENT_INVOKE_CURRENCY | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_CURRENCY event when it calls a popup currency calculator. Siebel Open UI makes this call in reply to a user action on the keyboard or mouse. You can use this event to configure Siebel Open UI to handle the open action that the currency calculator requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. The control object of the currency calculator that Siebel Open UI called. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebConsts.get("PHYEVENT_ INVOKE_CURRENCY"), function (control) { if (control === this.GetProperty("RevenueControl")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, can't open this currency field"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|---|
| PHYEVENT_INVOKE_TOGGLE | <p>Siebel Open UI triggers the PHYEVENT_INVOKE_TOGGLE event when it calls a control that includes a toggle layout configuration. Siebel Open UI makes this call in reply to a user action to toggle the layout. You can use this event to configure Siebel Open UI to handle the action that the toggle layout requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ value. Contains the value of the toggle control that exists after the user action. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebcConsts.get("PHYEVENT_ INVOKE_TOGGLE"), function (value) { if (value === this.GetProperty("FlaggedControl")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, change the Range value to toggle"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|-------------------------|---|
| PHYEVENT_DRILLDOWN_FORM | <p>Siebel Open UI triggers the PHYEVENT_DRILLDOWN_FORM event when it calls a drilldown control that resides on a form applet. Siebel Open UI makes this call in reply to a user click on the drilldown. You can use this event to configure Siebel Open UI to handle the action that the drilldown requests. You can use it with the following objects:</p> <ul style="list-style-type: none"> ■ control. Identifies the control object that contains the destination of the drilldown control. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebConsts.get("PHYEVENT_ DRILLDOWN_FORM"), function (control) { if (control === this.GetProperty("AccountDrill")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, change the Range value to drill down"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|--------------------------|--|
| PHYEVENT_ENTER_KEY_PRESS | <p>Siebel Open UI triggers the PHYEVENT_ENTER_KEY_PRESS event when the user presses the Enter key. Siebel Open UI triggers it only if a control is in focus. You can use this event to handle an Enter key press before the proxy layer uses the default configuration to handle it. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ control. Identifies the control object where the user used the Enter key. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(siebConsts.get("PHYEVENT_ENTER_KEY_PRESS"), function (control) { if (control === this.GetProperty("Salary")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, change the Range value to commit"); return (false); } } return (true); }); </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|---|
| PHYEVENT_ESC_KEY_PRESS | <p>Siebel Open UI triggers the PHYEVENT_ESC_KEY_PRESS event when the user presses the Esc (Escape) key. Siebel Open UI triggers it only if a control is in focus. You can use this event to handle an Esc key press before the proxy layer uses the default configuration to handle it. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ control. Identifies the control object where the user used the Esc key. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebelConsts.get("PHYEVENT_ESC_KEY_PRESS"), function (control) { if (control === this.GetProperty("Salary")){ var highValue = this.Get("HighVal"); if (highValue < this.Get ("MinRangeForFlagged")){ alert ("Sorry, change the Range value to undo"); return (false); } } return (true); });" </pre> |

Table 42. Events That You Can Use to Customize Form Applets

| Event | Description |
|------------------------|---|
| PHYEVENT_TAB_KEY_PRESS | <p>Siebel Open UI triggers the PHYEVENT_TAB_KEY_PRESS event when the user presses the Tab key. Siebel Open UI triggers it only if a control is in focus. You can use this event to handle a Tab key press before the proxy layer uses the default configuration to handle it. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ control. Identifies the control object where the user used the Tab key. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(siebeConsts.get("PHYEVENT_TAB_KEY_PRESS"), function (control) { if (control === this.GetProperty("Salary")){ var highValue = this.Get("HighVal"); if (highValue < this.Get("MinRangeForFlagged")){ alert ("Sorry, change the Range value to undo"); return (false); } } return (true); }); </pre> |

Events That You Can Use to Customize List Applets

[Table 43](#) describes the events that you can use to customize a list applet.

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|---------------------|--|
| PHYEVENT_SELECT_ROW | <p>Siebel Open UI triggers the PHYEVENT_SELECT_ROW event when the user chooses a row in a list applet. You can use it to determine whether or not the user clicked a row that is not the current row. Using this event might cause the state of objects that reside on the Siebel Server to be inconsistent with the state of objects that reside in the client.</p> <p>You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowIndex. Contains the index of the row that the user clicks. It uses 0 as the index for the first row. ■ shiftKey. Contains a Boolean value that indicates if the user pressed the Shift key while choosing a row. ■ controlKey. Contains a Boolean value that indicates if the user pressed the Ctrl key while choosing a row. <p>The following code is an example usage of this event:</p> <pre>"this.AttachEventHandler(siebConsts.get("PHYEVENT_SELECT_ROW"), function (rowIndex, shiftKey, controlKey) { if (rowIndex === 0 && shiftKey){ alert ("Do not multiselect all rows."); return (false); } });"</pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|-----------------------|--|
| PHYEVENT_COLUMN_FOCUS | <p>Siebel Open UI triggers the PHYEVENT_COLUMN_FOCUS event when a column in a list applet comes into focus. You can use it to identify the column that is in focus, and to call custom code. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowIndex. Contains the index of the current row. It uses 1 as the index for the first row. ■ control. Identifies the column control object that comes into focus. ■ value. Contains the value of the column control object. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebelConsts.get("PHYEVENT_COLUMN_FOCUS"), function (rowIndex, control, value) { if (rowIndex > 5) { return (true); } if (this.Get("AppletInFocus")){ var controlArray = this.Get("FlaggedControlSet"); if (controlArray.indexOf(control) >= 0){ // Declare the flagged control. var valueObject = this.GetProperty ("FlaggedControlValue"); if (value > maxVal ue){ // Flag value that is higher than allowed when receiving focus. this.SetProperty ("FlagHigher", true); } } } return (true); });" </pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|----------------------|--|
| PHYEVENT_COLUMN_BLUR | <p>Siebel Open UI triggers the PHYEVENT_COLUMN_BLUR event when a column in a list applet goes out of focus. You can use it to identify the column that is going out of focus, and to call custom code. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowIndex. Contains the index of the current row. It uses 1 as the index for the first row. ■ control. Identifies the column control object that is going out of focus. ■ value. Contains the value of the column control object. <p>The following code is an example usage of this event:</p> <pre> "this.AttachEventHandler(siebelConsts.get("PHYEVENT_COLUMN_BLUR"), function (rowIndex, control, value) { if (rowIndex > 5) { return (true); } if (this.Get("AppletInFocus")){ var controlArray = this.Get("FlaggedControlSet"); if (controlArray.indexOf(control) >= 0) { // This is a flagged control. var valueObject = this.GetProperty ("FlaggedControl sValue"); if (value > maxVal ue){ // Value higher than allowed when receiving focus. Let's flag this. this.SetProperty ("FlagH igher", true); } } } return (true); });" </pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|-------------------------|--|
| PHYEVENT_DRILLDOWN_LIST | <p>Siebel Open UI triggers the PHYEVENT_DRILLDOWN_LIST event when the user clicks a drilldown link in a list applet. You can use it to call custom code when the user clicks a drilldown link. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowIndex. Contains the index of the current row. It uses 1 as the index for the first row. ■ colName. Contains the name of the column where the drilldown link resides. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(siebConsts.get("PHYEVENT_DRILLDOWN_LIST"), function (colName, rowIndex) { if (name === "Type"){ var maxOptyArray = this.Get("m0"); if (maxOptyArray[rowIndex] > this.Get("HigVal")){ alert ("Fix opportunity value before drill down."); return (false); } } return (true); });" </pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|-----------------------|--|
| PHYEVENT_VSCROLL_LIST | <p>Siebel Open UI triggers the PHYEVENT_VSCROLL_LIST event when the user clicks the next page or prior page control in a list applet or tile applet. You can use it to call custom code when the user clicks one of these controls. Siebel Open UI does not trigger this event if the user uses a keyboard shortcut to do the pagination. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ direction. Includes one the following values that describes the type of pagination that the user attempted: <ul style="list-style-type: none"> ■ PAG_NEXT_RECORD. User paginated to the next record. ■ PAG_PREV_RECORD. User paginated to the previous record. ■ PAG_NEXT_SET. User paginated to the next set of record. ■ PAG_PREV_SET. User paginated to the previous set of record. ■ PAG_SCROLL_UP. User scrolled up. ■ PAG_SCROLL_DN. User scrolled down. <p>The following code is an example usage of this event:</p> <pre>"this.AttachEventHandler(siebConsts.get("PHYEVENT_VSCROLL_LIST"), function (direction) { if (direction ===SiebelApp.Contants.Get("PAG_NEXT_SET")){ alert ("Jump record by record. Not sets."); return (false); } return (true); });"</pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|--------------------|--|
| PHYEVENT_SORT_LIST | <p>Siebel Open UI triggers the PHYEVENT_SORT_LIST event when the user sorts a list column. You can use it to call custom code when the user does this sort. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ colName. Identifies the name of the column that the user is sorting. ■ direction. Identifies one of the following directions: <ul style="list-style-type: none"> ■ SORT_ASCENDING. The user is sorting the column in ascending order. ■ SORT_DESCENDING. The user is sorting the column in descending order. <p>The following code is an example usage of this event:</p> <pre> this.AttachEventHandler(sieConstants.get("PHYEVENT_SORT_LIST"), function (colName, direction) { if (colName === "Type"){ if (direction === SiebelApp.Constants.Get("SORT_ASCENDING")) { alert ("You cannot sort this column."); return (false); } } return (true); }); </pre> |

Table 43. Events That You Can Use to Customize List Applets

| Event | Description |
|------------------------|--|
| PHYEVENT_HIER_EXPAND | <p>Siebel Open UI triggers the PHYEVENT_HIER_EXPAND event when the user expands a row in a hierarchal list applet. You can use it to call custom code when the user does this expansion. Siebel Open UI uses this event only with hierarchal list applets. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowNum. Contains the row number that the user is expanding. It uses 0 as the number for the first row. <p>The following code is an example usage of this event:</p> <pre>"this.AttachEventHandler(siebConsts.get("PHYEVENT_HIER_EXPAND"), function (rowNum) { if (rowNum === 0){ var highValue = this.Get("HighVal"); if (highValue < this.Get("MinRangeForFlagged")){ alert("Sorry. Change the Range value to expand this row."); return (false); } } return (true); });"</pre> |
| PHYEVENT_HIER_COLLAPSE | <p>Siebel Open UI triggers the PHYEVENT_HIER_COLLAPSE event when the user collapses a row in a hierarchal list applet. You can use it to call custom code when the user does this collapse. Siebel Open UI uses this event only with hierarchal list applets. You can use this event with the following objects:</p> <ul style="list-style-type: none"> ■ rowNum. Contains the row number that the user is collapsing. It uses 0 as the number for the first row. <p>The following code is an example usage of this event:</p> <pre>"this.AttachEventHandler(siebConsts.get("PHYEVENT_HIER_COLLAPSE"), function (rowNum) { if (rowNum === 2){ var highValue = this.Get("HighVal"); if (highValue < this.Get("MinRangeForFlagged")){ alert("Sorry, change the Range to collapse this row."); return (false); } } return (true); });"</pre> |

Internationalization Support

Table 44 and Table 45 describe the locales that Siebel Open UI supports. It supports the i18N internationalization standard. It does not hard code strings and it uses language independent values for each LOV (list of values) instead of translated values.

Table 44. Locales That Siebel Open UI Supports for Windows, AIX, Oracle Solaris, and HP-UX

| Lang | Codepage | Windows | AIX | Oracle Solaris | HP-UX |
|------|----------|-----------------------|-------------|------------------|------------|
| CHS | 936 | Chinese, Simplified | ZH_CN.UTF-8 | zh_CN.UTF-8 | zh_CN.utf8 |
| CHT | 950 | Chinese, Traditional | ZH_TW.UTF-8 | zh_TW.UTF-8 | zh_TW.utf8 |
| CSY | 1250 | Czech | CS_CZ.UTF-8 | cs_CZ.UTF-8 | univ.utf8 |
| DAN | 1252 | Danish | DA_DK.UTF-8 | da_DK.UTF-8 | univ.utf8 |
| DEU | 1252 | German, Standard | DE_DE.UTF-8 | de_DE.UTF-8@euro | de_DE.utf8 |
| ENU | 1252 | English, American | EN_US.UTF-8 | en_US.UTF-8 | univ.utf8 |
| ESN | 1252 | Spanish, Modern | ES_ES.UTF-8 | es_ES.UTF-8@euro | es_ES.utf8 |
| FIN | 1252 | Finnish | FI_FI.UTF-8 | fi_FI.UTF-8@euro | univ.utf8 |
| FRA | 1252 | French, Standard | FR_FR.UTF-8 | fr_FR.UTF-8@euro | fr_FR.utf8 |
| ITA | 1252 | Italian, Standard | IT_IT.UTF-8 | it_IT.UTF-8@euro | it_IT.utf8 |
| JPN | 932 | Japanese | JA_JP.UTF-8 | ja_JP.UTF-8 | ja_JP.utf8 |
| KOR | 949 | Korean | KO_KR.UTF-8 | ko_KR.UTF-8 | ko_KR.utf8 |
| NLD | 1252 | Dutch, Standard | NL_NL.UTF-8 | nl_NL.UTF-8@euro | univ.utf8 |
| PTB | 1252 | Portuguese, Brazilian | PT_BR.UTF-8 | pt_BR.UTF-8 | univ.utf8 |
| PTG | 1252 | Portuguese, Standard | PT_PT.UTF-8 | pt_PT.UTF-8@euro | univ.utf8 |
| SVE | 1252 | Swedish | SV_SE.UTF-8 | sv_SE.UTF-8 | sv_SE.utf8 |
| THA | 874 | Thai, Thailand | TH_TH.UTF-8 | th_TH.UTF-8 | th_TH.utf8 |
| RUS | 1251 | Russian | RU_RU.UTF-8 | ru_RU.UTF-8 | ru_RU.utf8 |

Table 44. Locales That Siebel Open UI Supports for Windows, AIX, Oracle Solaris, and HP-UX

| Lang | Codepage | Windows | AIX | Oracle Solaris | HP-UX |
|------|----------|---------|-------------|----------------|------------|
| TRK | 1254 | Turkish | TR_TR.UTF-8 | tr_TR.UTF-8 | tr_TR.utf8 |
| POL | 1250 | Polish | PL_PL.UTF-8 | pl_PL.UTF-8 | pl_PL.utf8 |

Table 45. Locales That Siebel Open UI Supports for Linux RH, Linux SuSe, Enterprise Linux, and Java Locale Code

| Lang | Codepage | Linux RH | Linux SuSe | Enterprise Linux | Java Locale Code |
|------|----------|------------|------------|------------------|------------------|
| CHS | 936 | zh_CN.utf8 | zh_CN.utf8 | zh_CN.utf8 | zh_CN |
| CHT | 950 | zh_TW.utf8 | zh_TW.utf8 | zh_TW.utf8 | zh_TW |
| CSY | 1250 | cs_CZ.utf8 | cs_CZ.utf8 | cs_CZ.utf8 | cs_CZ |
| DAN | 1252 | da_DK.utf8 | da_DK.utf8 | da_DK.utf8 | da_DK |
| DEU | 1252 | de_DE.utf8 | de_DE.utf8 | de_DE.utf8 | de_DE |
| ENU | 1252 | en_US.utf8 | en_US.utf8 | en_US.utf8 | en_US |
| ESN | 1252 | es_ES.utf8 | es_ES.utf8 | es_ES.utf8 | es_ES |
| FIN | 1252 | fi_FI.utf8 | fi_FI.utf8 | fi_FI.utf8 | fi_FI |
| FRA | 1252 | fr_FR.utf8 | fr_FR.utf8 | fr_FR.utf8 | fr_FR |
| ITA | 1252 | it_IT.utf8 | it_IT.utf8 | it_IT.utf8 | it_IT |
| JPN | 932 | ja_JP.utf8 | ja_JP.utf8 | ja_JP.utf8 | ja_JP |
| KOR | 949 | ko_KR.utf8 | ko_KR.utf8 | ko_KR.utf8 | ko_KR |
| NLD | 1252 | nl_NL.utf8 | nl_NL.utf8 | nl_NL.utf8 | nl_NL |
| PTB | 1252 | pt_BR.utf8 | pt_BR.utf8 | pt_BR.utf8 | pt_BR |
| PTG | 1252 | pt_PT.utf8 | pt_PT.utf8 | pt_PT.utf8 | pt_PT |
| SVE | 1252 | sv_SE.utf8 | sv_SE.utf8 | sv_SE.utf8 | sv_SE |
| THA | 874 | th_TH.utf8 | th_TH.utf8 | th_TH.utf8 | th_TH |
| RUS | 1251 | ru_RU.utf8 | ru_RU.utf8 | ru_RU.utf8 | ru_RU |
| TRK | 1254 | tr_TR.utf8 | tr_TR.utf8 | tr_TR.utf8 | tr_TR |
| POL | 1250 | pl_PL.utf8 | pl_PL.utf8 | pl_PL.utf8 | pl_PL |

Screens and Views That Siebel Mobile Uses

This topic describes screens and views that Siebel Mobile uses. It includes the following information:

- [Screens and Views That Siebel Consumer Goods Uses on page 590](#)
- [Screens and Views That Siebel Sales Uses on page 591](#)
- [Screens and Views That Siebel Service Uses on page 593](#)
- [Screens and Views That Siebel Pharma Uses on page 594](#)

Screens and Views That Siebel Consumer Goods Uses

[Table 46](#) lists the predefined screens and views that Siebel Mobile uses for Siebel Consumer Goods.

Table 46. Screens and Views That Siebel Consumer Goods Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|----------|----------------------|--|
| Accounts | Addresses | CG Account Addresses View - Mobile |
| | Calls | CG Account Calls Views - Mobile |
| | Contacts | CG Account Contacts View - Mobile |
| | Accounts | CG Account List View - Mobile |
| | Notes | CG Account Notes View - Mobile |
| | Orders | CG Account Orders View - Mobile |
| | Retail Audits | CG Account Product Audits View - Mobile |
| | Product Distribution | CG Account Products View - Mobile |
| Contacts | Addresses | CG Contact Addresses View - Mobile |
| | Best Call Time | CG Contact Best Call Times View - Mobile |
| | Contacts | CG Contact List View - Mobile |
| Routes | Route Accounts | CG Routes Accounts View - Mobile |
| | Routes | CG Routes List View - Mobile |

Table 46. Screens and Views That Siebel Consumer Goods Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|---------|----------------------|---|
| Calls | Assessment | CG Call Account Assessment View - Mobile |
| | Notes | CG Call Account Notes View - Mobile |
| | Merchandising Audits | CG Call Merchandising Audits View - Mobile |
| | Orders | CG Call Orders View - Mobile |
| | Retail Audits | CG Call Retail Audit List View - Mobile |
| | Calls | CG Outlet Visit Activities List View - Mobile |
| | Call Items | CG Visit Call Items List View - Mobile |
| Orders | Orders | CG Order List View - Mobile |
| Returns | Returns | CG Return Order List View - Mobile |

Screens and Views That Siebel Sales Uses

Table 47 lists the predefined screens and views that Siebel Mobile uses for Siebel Sales.

Table 47. Screens and Views That Siebel Sales Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|----------|-----------------------|--|
| Accounts | Accounts | SHCE Account List View - Mobile |
| | Account Contacts | SHCE Account Contacts View - Mobile |
| | Account Opportunities | SHCE Account Opportunity View - Mobile |
| | Account Address | SHCE Account Address View - Mobile |
| | Account Activities | SHCE Account Activities View - Mobile |
| | Account Team | SHCE Account Team View - Mobile |
| Contacts | Contacts | SHCE Sales Contact List View - Mobile |
| | Contact Opportunities | SHCE Sales Contact Opportunities View - Mobile |
| | Contact Team | SHCE Contact Team View - Mobile |
| | Contact Addresses | SHCE Contact Address View - Mobile |
| Leads | Leads | SHCE Sales Lead List View - Mobile |
| | Lead Opportunities | SHCE Sales Lead Opportunities View - Mobile |

Table 47. Screens and Views That Siebel Sales Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|---------------|------------------------|---|
| Opportunities | Opportunities | SHCE Opportunities List View - Mobile |
| | Opportunity Contacts | SHCE Sales Opportunities Contacts View - Mobile |
| | Opportunity Products | SHCE Sales Opportunities Products View - Mobile |
| | Opportunity Quotes | SHCE Sales Opportunities Quotes View - Mobile |
| | Opportunity Activities | SHCE Sales Opportunities Activities View - Mobile |
| | Opportunity Team | SHCE Sales Opportunities Opportunity Team View - Mobile |
| Quotes | Quotes | SHCE Quote List View - Mobile |
| | Quote Items | SHCE Quote QuoteItem View - Mobile |
| | Quote Orders | SHCE Quote Order View - Mobile |
| | Quote Team | SHCE Quote Team View - Mobile |
| Orders | Orders | SHCE Sales Orders List View - Mobile |
| | Order Items | SHCE Sales Order line Item View - Mobile |
| Activities | Activities | SHCE Activity List View - Mobile |
| | Activity Contact | SHCE Sales Activity Contact Form View - Mobile |
| | Activity Employee | SHCE Sales Activity Employee Form View - Mobile |

Screens and Views That Siebel Service Uses

Table 48 lists the predefined screens and views that Siebel Mobile uses for Siebel Service.

Table 48. Screens and Views That Siebel Service Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|------------------|----------------------------|---|
| Activities | Service Activities | SHCE Service Activity Home Page View - Mobile |
| | Activity Contacts | SHCE Service Activity Contact Form View - Mobile |
| | Activity Recommended Part | SHCE Service FS Activity Recommended Parts Tools - Mobile |
| | Activity Steps | SHCE Service Activity FS Steps View - Mobile |
| | Activity Instructions | SHCE Service Activity FS Instructions List view - Mobile |
| | Activity Part Tracker | SHCE Service FS Activity Part Movements View - Mobile |
| | Activity Time Tracker | SHCE Service Activity Time View - Mobile |
| | Activity Expense Tracker | SHCE Service Activity FS Expense View - Mobile |
| | Activity Invoices | SHCE Service FS Invoice - Auto Invoice View - Mobile |
| Service Requests | Service Requests | SHCE Service Service Request View - Mobile |
| | Service Request Orders | SHCE Service SR Orders View - Mobile |
| | Service Request Activities | SHCE Service SR Activity View - Mobile |
| | Service Request Invoices | SHCE Service SR Invoices View - Mobile |
| Accounts | Accounts | SHCE Service Accounts View - Mobile |
| | Account Contacts | SHCE Service Account Contacts View - Mobile |
| | Account Service Requests | SHCE Service Account SRs View - Mobile |
| | Account Assets | SHCE Service Account Assets View - Mobile |
| | Account Entitlements | SHCE Service Account Entitlements View - Mobile |
| Browser | Part Browser | SHCE Service My Part Browser View - Mobile |
| | Part Browser Availability | SHCE Service Part Browser Availability View - Mobile |
| | Part Browser Substitutes | SHCE Service Part Browser Substitute View - Mobile |

Table 48. Screens and Views That Siebel Service Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|----------|---------------------|---|
| Orders | Orders | SHCE Service Orders List View - Mobile |
| | Order Line Items | SHCE Service Order line Item View - Mobile |
| Invoices | Invoices | SHCE Service Invoice List View - Mobile |
| | Invoice Line Items | SHCE Service Invoice line Item View - Mobile |
| Assets | Assets | SHCE Service Asset List View - Mobile |
| | Asset Measurements | SHCE Service Asset Measurement View - Mobile |
| | Asset Warranty | SHCE Service Asset Warranty View - Mobile |
| | Asset Entitlements | SHCE Service Asset Entitlements View - Mobile |
| | Asset Readings | SHCE Service Asset Reading View - Mobile |

Screens and Views That Siebel Pharma Uses

Table 49 lists the predefined screens and views that Siebel Mobile uses for Siebel Pharma.

Table 49. Screens and Views That Siebel Pharma Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|----------|---------------------|---|
| Accounts | Addresses | Pharma Account Addresses View - Mobile |
| | Affiliations | Pharma Account Affiliations View - Mobile |
| | Calls | Pharma Account Calls View - Mobile |
| | Contacts | Pharma Account Contact View - Mobile |
| | Accounts | Pharma Account List View - Mobile |
| | Relationships | Pharma Account Relationships View - Mobile |
| Contacts | Addresses | Pharma Contact Address View - Mobile |
| | Affiliations | Pharma Contact Affiliations View - Mobile |
| | Best Time | Pharma Contact Best Contact Times View - Mobile |
| | Calls | Pharma Contact Call View - Mobile |
| | Contacts | Pharma Contact List View - Mobile |
| | Relationships | Pharma Contact Relationships View - Mobile |
| | State Licenses | Pharma Contact State Licenses View - Mobile |

Table 49. Screens and Views That Siebel Pharma Uses

| Screen | View Name in Client | View Name in Siebel Tools |
|--------|---------------------------|---|
| Calls | Attendees | SIS HH Pharma Account Call Attendee View - Mobile |
| | Product Details | SIS HH Pharma Professional Call Products Detailed View - Mobile |
| | Promotional Items Dropped | SIS HH Pharma Professional Promotional Items View - Mobile |
| | Samples Dropped | SIS HH Pharma Professional Samples Dropped View - Mobile |
| | Calls | LS Pharma Professional Call Execute View - Mobile |
| | (No Title) | LS Pharma Call Signature Capture View - Mobile |
| | Validation Results | LS Pharma Call Validation Results View - Mobile |

Controls That Siebel Open UI Uses

This topic describes the controls that Siebel Open UI uses.

Predefined Controls That Siebel Open UI Uses

Table 50 describes the predefined controls that Siebel Open UI uses.

Table 50. Predefined Controls That Siebel Open UI Uses

| Control Name | Description | Where Defined |
|--------------|--|-------------------|
| Currency | Sets the currency. | controlbuilder.js |
| DetailPopup | Displays more information for a field. | |
| EffDat | Sets the effective date. | |
| Mvg | Chooses more than one value. | |
| PhoneCtrl | Enters a phone number. | |
| Pick | Chooses a value. | |
| SelectCtrl | Makes a selection. | |
| List UI | Displays records in a list. | jqgridrenderer.js |

Table 50. Predefined Controls That Siebel Open UI Uses

| Control Name | Description | Where Defined |
|---------------|---|-----------------------|
| btnControl | Displays a button. | phyrenderer.js |
| chartControl | Displays a chart. | |
| checkBoxCtrl | Displays a checkbox. | |
| comboControl | Displays a combobox. | |
| fileControl | Uploads files. | |
| imageControl | Displays an image. | |
| ink | Captures a signature in a mobile environment. | |
| label | Displays a label. | |
| link | Displays a link that navigates to another view. | |
| mailtoControl | Sends an email message. | |
| plainText | Displays the contents of a field that is not editable, is not navigable, or does not possess a state. For example, a label. | |
| radioButton | Displays a radio button. | |
| text | Displays text. | |
| textArea | Displays a text area. | |
| urlControl | Displays an external URL. For example, http://www.google.com . | |
| Map UI | Displays a map. | siebelmaprender.js |
| Tiles UI | Displays records in a tile. | Tilesrollcontainer.js |

Other Controls That Siebel Open UI Uses

Table 51 describes other controls that Siebel Open UI uses. Siebel Open UI uses all these controls in the controlbuilder.js file in addition to the files that the Where Defined column lists.

Table 51. Other Controls That Siebel Open UI Uses

| Control Name | Description | Where Defined |
|--------------|-------------------------|-------------------------------|
| DatePick | Sets the date. | datepicker-ext.js |
| DateTimePick | Sets the date and time. | jquery-ui-timepicker-addon.js |
| Calculator | Displays a calculator. | jquery.calculator.zip |

Table 51. Other Controls That Siebel Open UI Uses

| Control Name | Description | Where Defined |
|--------------|-------------------|---|
| RTCEditor | Enters rich text. | ckeditor_3.6.3.zip ckeditor-custom.zip |
| FileUploader | Uploads files. | jquery.fileupload.js |

Mobile Controls That Siebel Open UI Uses

Table 52 describes the mobile controls that Siebel Open UI uses. All controls are predefined controls except for the swipeButton and mobiscroll controls, which Siebel Open UI might modify slightly.

Table 52. Mobile Controls That Siebel Open UI Uses

| Control Name | Description | Where Defined |
|-------------------------------|---|---|
| JQMCaListRenderer | Sets the date in an activity list. | jqmcallistrenderer.js |
| JQMFormRenderer | Displays a form. | jqmformrenderer.js |
| JQMGridRenderer | Displays a grid. | jqmgridrenderer.js |
| JQMNavBarRenderer | Displays the screen navigation bar. | jqmjsNavBar.js, jqmnavbasepr.js |
| JQMLayout | Displays a layout transition. | jqmlayout.js |
| JQMListRenderer | Displays a list. | jqmlistrenderer.js |
| JQMMapCtrl | Displays a Google map. | jqmmapctrl.js |
| MBMenuRenderer | Displays a popup menu. | jqmmbmenurenderer.js |
| JQMPDQPhyRenderer | Displays a predefined query. | jqmpdqphyrenderer.js |
| JQMScrollContainer | Displays a scrolling list. | jqmscrollcontainer.js |
| JQMSearchCtrl | Displays a search control. | jqmsearchctrl.js |
| JQMLaunchpadNavRend er | Displays the Site map. | jqmsitemaprenderer.js, jqmnavbasepr.js |
| JQMTabletGridRenderer | Displays a grid in a tablet. | jqmtabletgridrenderer.js |
| JQMTabletListRenderer | Displays a list in a tablet. | jqmtabletlistrenderer.js |
| JQMToolbarRenderer | Displays a toolbar. | jqmtoolbarrenderer.js |
| JQMVisDropdownPhyRen derer | Displays a dropdown list for visibility. | jqmvisibilitydropdownprenderer.js |
| swipeButton | Displays the swipe button. | jquery.swipeButton.min.js |
| mobiscroll | Displays a scroller, such as the DateTime scroller. | mobiscroll.custom-2.5.0.min.js |

Browser Script Compatibility

Siebel Open UI supports your existing browser script. However, it is recommended that you customize a presentation model instead of using browser script. It is recommended that you gradually move any logic that you implement through your existing browser script to the presentation model.

You can write a browser script in JavaScript. This script can interact with the Document Object Model (DOM) and with the Siebel Object Model that is available in the Web browser through a shadow object. You can script the behavior of Siebel events and the browser events that the DOM exposes.

Siebel Open UI uses a JavaScript environment that allows you to implement browser scripting. This JavaScript API can dynamically refresh page content and instantly commit customization modifications. If your implementation currently uses browser scripting, then you can refactor JavaScript to move from your existing employee application to Siebel Open UI. *Refactoring* is the process of modifying the internal structure of existing code without modifying the external behavior of this code. For more information about this JavaScript API, see [Appendix A, "Siebel Open UI Application Programming Interface."](#)

Sequence That Siebel Open UI with Custom Browser Script

The following pseudocode describes the sequence that Siebel Open UI uses if your deployment includes custom browser script:

```
PR calls a PM method or event
  PM method or event calls an applet proxy method
    applet proxy method calls Applet_PreInvokeMethod on browser script
    applet proxy uses Call-Server to run applet method on Siebel Server
  PM runs Attach Pre Proxy binding for this applet method
  applet proxy calls Applet_InvokeMethod that resides in browser script
  PM runs Attach Post Proxy binding for this applet method
  applet proxy method ends
  PM method or event ends
PR call ends
```

where:

- PR is the physical renderer
- PM is the presentation model

For example, the following pseudocode describes the sequence that Siebel Open UI uses if the user clicks New in an applet, and if your deployment includes custom browser script that uses a method named NewRecord:

```
PR calls PM.OnControlEvent
  PM.OnControlEvent calls Applet_InvokeMethod
    Applet_InvokeMethod calls BrowserScript.Applet_PreInvokeMethod
    Applet_InvokeMethod calls Siebel Server to run NewRecord method on applet
  PM calls PM.AttachPreProxyExecuteBinding for the NewRecord method
  Applet_InvokeMethod calls BrowserScript.Applet_InvokeMethod
  PM calls PM.AttachPostProxyExecuteBinding for the NewRecord method
```

```

    Applet.InvokeMethod ends
    PM.OnControlEvent ends
    PR call ends

```

How Siebel Open UI Handles Custom Client Scripts

Siebel Open UI uses browser script through a JavaScript *shadow object*, which is a type of object that Siebel Open UI uses for client scripting. All other client objects include a corresponding shadow object, except for the PropertySet. For example, the JSSApplet object includes the JSSAppletShadow shadow object. Siebel Open UI exposes this shadow object to scripting. When Siebel Open UI prepares to display the applet, SWE determines whether or not a browser script is defined for this applet. If this script exists, then Siebel Open UI downloads the browser script file that contains the definition of the shadow object from the Siebel Server to the client.

For example, assume you write a browser script for an applet to handle the PreInvokeMethod event. At run-time, Siebel Open UI creates a JavaScript object that it derives from the JSSAppletShadow object. It runs the PreInvokeMethod event and the event handler of the shadow object before it calls the DoInvokeMethod event. Each shadow object includes a reference to the underlying object. The shadow object sends the call to this underlying object, if necessary. For more information about deriving values, see [“About Using This Book” on page 30](#).

How Siebel Open UI Creates Shadow Objects for Applications

Siebel Open UI creates an application shadow object with the following application object during application startup:

```

Application.InvokeMethod
.....
bRet = this.*FirePreInvokeMethod*(methodName, inputPS);
;return from here if the return value of the PreInvokeMethod is CancelOperation
; continue to invokeMethod if the return value is ContinueOperation
.....
this.DoInvokeMethod (methodName, args);
this.*FireInvokeMethod*(methodName, inputPS);

```

How Siebel Open UI Creates Shadow Objects for Business Objects

Siebel Open UI uses a business object shadow object only in other shadow objects, such as an application shadow object, applet shadow object, or business component shadow object.

How Siebel Open UI Creates Shadow Objects for Applets, Business Components, or Business Services

Siebel Open UI does the following to create a shadow object for an applet, business component, or business service:

- **Siebel Server.** Siebel Open UI creates the ObjInfo (SWE_PROP_SHADOW) shadow when it encounters an object that includes a custom script that you write. The server gets the class name and the file name of the shadow from the SRF. It packages the class name and file name into the SWE_PROP_SHADOW, and then sends it to client.

- **Client.** Siebel Open UI gets the class name and the file name from SWE_PROP_SHADOW, loads the script file, creates the shadow object with the retrieved class name, and then stores the shadow pointer in the applet object or the business component object.

The process is the same for a business service except Siebel Open UI uses the SWE_PST_SERVICE_SHADOWS shadow.

How Siebel Open UI Creates Shadow Objects for Controls

Siebel Open UI does the following to create a shadow object for a control:

- **Siebel Server.** Siebel Open UI creates the ObjInfo (SWE_PROP_SHADOW) shadow object if it encounters a control that includes a script that you write. It gets the event name of the control from the SRF. It packages event names into the SWE_PST_SCRIPTS shadow, and then sends it to client.
- **Client.** Siebel Open UI gets the list of control events from the SWE_PST_SCRIPTS shadow, and then calls these methods from the corresponding predefined methods.

Browser Script Object Types

You can use the following object types in browser script:

- Application
- Applet
- Control
- Business object
- Business component
- Business service property

Event Handlers You Can Use to Handle Predefined Events

Table 53 describes the event handlers that you can use in browser script to handle a predefined event for a Siebel object type.

Table 53. Event Handlers You Can Use in Browser Script for a Siebel Object Type

| Object Type | Event Handler |
|--------------------|---|
| Application | <p>You can use the following event handlers:</p> <ul style="list-style-type: none"> ■ Application_InvokeMethod ■ Application_PreInvokeMethod |
| Applet | <p>You can use the following event handlers:</p> <ul style="list-style-type: none"> ■ Applet_ChangeFieldValue ■ Applet_ChangeRecord ■ Applet_InvokeMethod ■ Applet_PreInvokeMethod ■ Applet_Load |
| Business component | <p>You can use the following event handler:</p> <ul style="list-style-type: none"> ■ BusComp_PreSetFieldValue |
| Business service | <p>You can use the following event handlers:</p> <ul style="list-style-type: none"> ■ Service_InvokeMethod ■ Service_PreCanInvokeMethod ■ Service_PreInvokeMethod |

Event Handlers You Can Use to Handle Predefined DOM Events

Table 54 describes the event handlers that you can use in browser script to handle a predefined DOM event for a Siebel control object type.

Table 54. DOM Event Handlers You Can Use in Browser Script for a Siebel Object Type

| Object Type | Event Handler |
|---|--|
| Control in a high-interactivity client. | <p>You can use the following event handlers:</p> <ul style="list-style-type: none"> ■ OnBlur ■ OnFocus |
| Control in a standard interactivity client. | <p>You can use the following event handlers:</p> <ul style="list-style-type: none"> ■ onfocus ■ onblur ■ onmouseover ■ onmouseout ■ onclick ■ onchange |

Methods You Can Use in Browser Script

Table 55 describes the methods that you can use in a browser script for each Siebel object type that Oracle's Siebel Open UI can use.

Table 55. Methods You Can Use in Browser Script for Each Siebel Object Type

| Object Type | Method |
|-------------|--|
| Applet | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ ActiveMode ■ BusComp ■ BusObject ■ FindActiveXControl ■ FindControl ■ InvokeMethod ■ Name ■ ReInit |
| Application | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ FindApplet ■ ActiveApplet ■ ActiveViewName ■ ActiveBusObject ■ ActiveBusComp ■ FindBusObject ■ GetProfileAttr ■ GetService ■ InvokeMethod ■ IsReady ■ Name ■ NewPropertySet ■ SWEAlert ■ ShowModalDialog ■ SeblTrace |

Table 55. Methods You Can Use in Browser Script for Each Siebel Object Type

| Object Type | Method |
|--------------------|--|
| Business Component | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ BusObject ■ GetFieldValue ■ GetFormattedFieldValue ■ GetSearchExpr ■ GetSearchSpec ■ InvokeMethod ■ Name ■ SetFieldValue ■ SetFormattedFieldValue ■ WriteRecord |
| Business Object | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ FirstBusComp ■ GetBusComp ■ Name ■ NextBusComp |
| Business Service | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ InvokeMethod ■ Name ■ SetProperty ■ PropertyExists ■ RemoveProperty ■ GetProperty ■ GetFirstProperty ■ GetNextProperty |

Table 55. Methods You Can Use in Browser Script for Each Siebel Object Type

| Object Type | Method |
|-------------|---|
| Control | <p>You can use the following methods:</p> <ul style="list-style-type: none">■ Applet■ BusComp■ GetValue■ Name■ SetValue■ SetReadOnly■ SetEnabled■ SetVisible■ SetProperty■ GetLabelProperty■ GetProperty■ SetLabelProperty |

Table 55. Methods You Can Use in Browser Script for Each Siebel Object Type

| Object Type | Method |
|--------------|---|
| Property Set | <p>You can use the following methods:</p> <ul style="list-style-type: none"> ■ AddChild ■ Copy ■ GetChild ■ GetChildCount ■ GetFirstProperty ■ GetNextProperty ■ GetProperty ■ GetPropertyCount ■ GetType ■ GetValue ■ InsertChildAt ■ PropertyExists ■ RemoveChild ■ RemoveProperty ■ Reset ■ SetProperty ■ SetType ■ SetValue |

Glossary

access control

The set of Siebel CRM mechanisms that control the records that the user can access and the operations that the user can perform on the records.

account

A financial entity that represents the relationships between a company and the companies and people with whom the company does business.

ActiveX

A loosely defined set of technologies developed by Microsoft for sharing information among different applications.

ActiveX control

A specific way to implement ActiveX technology. It denotes reusable software components that use the component object model (COM) from Microsoft. ActiveX controls provide functionality that is encapsulated and reusable to programs. They are typically, but not always, visual in nature.

activity

Work that a user must track. Examples include a to-do, email sent to a contact, or a calendar entry with a contact.

activity (Siebel CRM)

An object in the Action business component of the Siebel data model that organizes, tracks, and resolves a variety of work, from finding and pursuing an opportunity to closing a service request. An Activity also captures an event, such as scheduling a meeting or calendar entry that occurs at a specific time and displays in the calendar.

administrator

Anyone who uses an administrative screen in the client to configure Siebel CRM. The Administration - Server Configuration screen is an example of an administrative screen.

applet metadata

The applet object in the Siebel Repository File that contains information that Siebel Open UI uses to bind the user interface to the business component.

asynchronous request

A type of request that Siebel Open UI sends to the Siebel Server and then continues other processing while it waits for a reply to this request.

Also see [synchronous request](#).



business component

A logical representation of one or more Siebel tables that usually contains information for a particular functional area, such as opportunity, account, contact, or activity. A business component can be included in one or more business objects.

business object

A logical representation of CRM entities, such as accounts, opportunities, activities, and contacts, and the logical groupings and relationships among these entities. A business object uses links to group business components into logical units. The links provide the one-to-many relationships that govern how the business components interrelate in this business object. For example, the opportunity business object groups the opportunity, contact, and activities business components.

carouselrenderer.js file

The physical renderer that bridges the JCarousel 3rdParty Control plug-in to the list presentation model that the listmodel.js file defines.

client

The client of a Siebel business application. Siebel Call Center is an example of a Siebel business application. Siebel Open UI renders the user interface in this client.

client computer

The computer that the Siebel Open UI user uses.

client file

A file that Siebel Open UI uses on the client computer.

contact

A person with whom a user might be required to phone or email to pursue a selling relationship. Various business objects can refer to a contact, and this does not require a relationship between the customer and contact. In Siebel CRM, a contact attribute in the context of a business object is a party that might or might not have a relationship defined.

CRM (Customer Relationship Management)

A software application that helps a business track customer interactions.

customization

The process of modifying Siebel Open UI to meet the specific requirements of your organization.

derive

To calculate a value by using one or more properties as input. For example, Siebel CRM can derive the value of a physical renderer property from one or more other properties.

focus

Indicates the currently active object in the client. Siebel CRM typically sets the border of this object to a solid blue line to identify the object that is in focus. For example, if the user edits the value of field A, then Siebel CRM places a blue border around the perimeter of field A. If the user tabs from field A to field B in a record, then Siebel CRM removes the blue border from field A and adds a blue border to field B.

high interactivity

A user interface mode that resembles a Windows client. It supports fewer browsers than standard interactivity, but it includes a set of features that simplify data entry. For example, page refreshes do not occur as often as they do in standard interactivity. The user can create new records in a list, save the data, and then continue browsing without encountering a page refresh.

infinite scroll

A feature that allows the user to scroll through records in a list applet indefinitely.

inheritance chain

An object-oriented programming technique that Siebel Open UI uses where one class modifies another class.

jqmlistrenderer and jqmgridrenderer

Physical renderers that Siebel Open UI uses to render a list applet in Siebel Mobile.

JQM Grid Renderer

An object that uses the `jqmgridrenderer.js` file to render data in a grid in the client.

Manifest File

An XML file that identifies the JavaScript files that Siebel Open UI must download to the client browser. It maps applet user properties to JavaScript files. The Siebel Web Engine gets the JavaScript implementation file information from the Manifest File the first time a user accesses an applet during a user session. To get this information, it looks up the keys that match the user property values for this applet.

metadata

Object definitions in the Siebel repository that describe the current state of Siebel Open UI.

metadata files

XML files that hold information on how the user experience must be shaped. Siebel Open UI uses metadata files to perform field mapping with the user interface, look ups in the user interface, application object mapping, and general representation of the user interface.

native mode

A user interface mode that allows you to deploy Siebel Open UI to the native Siebel Open UI client.

object definitions

The metadata that Siebel Open UI uses to run a Siebel application. The Account List Applet that Siebel Tools displays in the Object List Editor is an example of an object definition. It includes metadata that Siebel Open UI uses to render the Account List Applet, such as the height and width of all controls in the applet, and all the text labels that it must display on these controls.

opportunity

A qualified sales engagement that represents potential revenue where a sales representative is willing to officially commit to the pipeline and to include revenue in the sales forecast. The sales representative monitors the opportunity life cycle. This representative might be compensated depending on the results of cumulative sales and potentially how well the representative maintains details about the opportunity.



object manager

A system manager that hosts a Siebel application, providing the central processing for HTTP transactions, database data, and metadata. The Siebel Web Engine and data manager operate as facilities in the object manager.

parent business component

A business component that provides the one in a one-to-many relationship between two business components in a parent-child relationship.

physical renderer

A JavaScript file that Siebel Open UI uses to build the user interface. It allows you to use custom or third-party JavaScript to render the user interface. It binds a presentation model to a physical control. It allows this presentation model to remain independent of the physical user interface objects layer.

physical renderer methods

Life cycle methods that you code into any renderer.

predefined Siebel Open UI

The ready-to-use version of Siebel Open UI that Oracle provides you before you make any customization to Siebel Open UI.

predefined object

An object that comes defined with Siebel CRM. The objects that Siebel Tools displays in the Object List Editor immediately after you install Siebel Tools and the SRF (Siebel Repository File), but before you make any customization are predefined objects.

Presentation Model

A JavaScript file that allows you to customize behavior, logic, and content in the client. It contains the metadata and data from the applet and business component that Siebel Open UI uses to render a simple list applet or form applet. It determines the logic to apply, captures client interactions, such as the user leaving a control, collects field values, and sets properties.

Presentation Model class

A class that includes life cycle methods that you code for the presentation model and control methods that Siebel Open UI uses to add presentation model properties and behavior.

private field

A type of field that only allows the record owner to view the record. For more information, see *Siebel Object Types Reference*.

proxy object

An object instance that Siebel Open UI uses for the client proxy.

responsibility

An entity in the Siebel data model that determines the views that the user can access. For example, the responsibility of the sales representative allows the user to access the My Opportunities view, whereas the responsibility of the Siebel CRM developer allows the user to access administration views. A Siebel CRM developer or system administrator defines the responsibilities.

shadow object

A type of object that Siebel Open UI uses for client scripting.

Siebel Business Application

An application that is part of Siebel CRM, such as Siebel Call Center.

Siebel CRM data

Business data that is created in Siebel Open UI, data that is created in the client of a Siebel Business Application, such as Siebel Call Center, or data that resides in the Siebel database on the Siebel Server. Examples include an opportunity, account, or activity.

Siebel Open UI

An open architecture that you can use to customize the user interface that your enterprise uses to display business process information.

Siebel Repository

A set of database tables that stores object definitions. Examples of types of objects include applets, views, business components, and tables. You use Siebel Tools to create or modify an object definition.

Siebel Repository File (SRF)

A flat file that resides on the Siebel Server. This file contains the object definitions that define a Siebel business application. You use Siebel Tools to modify this file.

Siebel Property Set

A hierarchy that Siebel Open UI uses to communicate between objects that reside on the Siebel Server and the proxies that reside in the client.

Siebel Server

The server that runs the Siebel Server software. The Siebel Server processes business logic and data access for Oracle's Siebel Open UI.

Siebel Web services

Provides access to an existing Siebel business service or workflow process as a Web service to be consumed by an external application.

synchronous request

A type of request that Siebel Open UI sends to the Siebel Server and then waits for a reply to this request before it continues any other processing. An asynchronous request is a type of request that Siebel Open UI sends to the Siebel Server and then continues other processing while it waits for a reply to this request.

Also see [asynchronous request](#).

standard interactivity

A user interface mode that resembles most traditional Web applications. It supports different types of browsers. Page refreshes occur often, such as if the user creates a new record, submits a form, or browses through a list of records.



SWE run-time applet object

An object that exposes scripting interfaces that allow you to modify the applet so that it can control the business component or business service that this applet references.

user

A person who uses the client of a Siebel business application to access Siebel CRM data.

user interface

The graphical user interface that the user uses in the client.

Index

A

ActiveX

- architecture 53
- Open UI usage of 28
- usage comparison 29

APIs

- guidelines for using to customize Presentation Model 97
- JavaScript description 598
- public JavaScript support 20
- usage with Physical Renderer during life cycle 54
- usage with Presentation Model and Physical Renderer in life cycle 54
- using to configure scrolling 337

applets

- customizing for indefinite scrolling 340
- customizing look and feel of 289
- customizing menus on 291
- customizing scrolling of 337
- customizing swipe in 342
- customizing to display maps 344
- displaying external data in 256
- displaying in an external application 272
- event handler usage with scripts 601
- example usage in object hierarchy 47
- example usage in object life cycle 59
- modifying to customize calendar event styles 209
- modifying to display a box list 164
- renderer usage with 276
- rendering as a carousel 167
- rendering as a carousel without compiling 173
- rendering as a grid 176
- support for collapsing 275, 353
- support for expanding, collapsing, sizing 19
- SWE run-time usage 47
- usage as scripting shadow object 599
- usage of metadata 47
- usage with proxy objects 47
- user property support for calendar days 212
- user property support for calendar timestamps 213

architecture 17, 21, 49, 51, 53, 611

- about Siebel Open UI 37
- differences between high interactivity and

- Siebel Open UI 21, 28
- differences in client architecture between high interactivity and Siebel Open UI 53
- differences in customization between high interactivity and Siebel Open UI 29
- differences in server architecture between high interactivity and Siebel Open UI 51
- example object hierarchy 47
- example of object life cycle 58
- life cycle of an element 54
- life cycle of user interface elements 54
- object hierarchy 46
- overview of Siebel Open UI development 37
- presentation model and physical renderer 39
- presentation model life cycle methods 54
- rendering 19

B

browser

- GPS support 515
- scripting 29, 98, 598
- standards compliance 17
- types supported 17

business components

- modifying to customize calendar event styles 209
- modifying to display Google map in list applet 349, 350
- modifying to display Google map in separate window 347
- modifying to remove buttons or icons 350
- usage as shadow object 599
- usage with applets 47
- usage with notification property set 98
- usage with proxy object 47
- using to display data in external applications 256

C

caches

- clearing in browser 204
- file organization in 122

cascading style sheets

- default usage 284
- event style usage example of 210

- guideline for usage 99
- mobile scroll view usage 339
- opening for editing 279
- organization of 122
- style usage for pick icon 489
- theme_base.css usage 166
- theme-black.css usage 277
- theme-blue.css usage 285
- theme-calendar.css usage 211
- third-party usage of 124
- usage compared to hard coding 29
- usage to control layout and styling 47
- using to layout div elements 28
- where stored 124

client

- multiple client environments 20

customization example

- adding a map that includes location data in a mobile application 343
- applying a Siebel Web Template to a mobile view 288
- configuring a list applet to render as a carousel 167
- configuring a list applet to render as a carousel without compiling the SRF 173
- configuring Siebel Open UI to partially refresh screens 159
- customizing a mobile list 299
- displaying desktop views in Mobile Web Clients 286
- displaying or hiding fields 203
- embedding Siebel views or applets in an external application 272
- integrating external application data in a Siebel view 253, 256
- modifying a list to display a box list 164

customizing mobile applications

- using application templates and stylesheets 277
- using third party JavaScript plugins 275, 276, 277

customizing the calendar

- controlling how the calendar displays timestamps 213
- customizing event styles 211
- customizing repeating calendar events for Mobile 212
- deploying calendars according to your calendar deployment requirements 208
- specifying the first day of the week 212
- specifying values for the work days and week start fields 212

- specifying work days 212

D

DOM

- access to 19
- events 28
- HTML structure for JavaScript control 29
- predefined event 602
- specifying element as JQuery object 489
- usage guidelines 99
- usage with script 598

E

email

- support with maps 514

G

guidelines

- configuring physical renderer 99
- configuring presentation model 97
- configuring presentation model and physical renderer for client object 100

H

high interactivity

- calendar support in Open UI 208
- comparison to Open UI 21
- displaying views 286
- file organization 124

HTML

- CLASS tag in 210
- div element 28, 29, 337, 338
- DOM structure 29
- event style tags in 210
- example of in swt file 283

I

image files

- customizing the background 147
- where Open UI stores them 124

J

JavaScript

- API 20, 598
- browser script usage 598
- example usage 168
- file usage 100, 124
- framework 598
- plug-ins 276
- rendering in objects 28
- usage for controlling logic 47
- using to determine if Siebel Open UI is

- enabled 278
- variable usage 98

jQuery

- calendar 48
- plug-ins for Mobile 276

M

Manifest File

- guideline for using with Presentation Model and Physical Renderer 100
- using to create custom lists 299
- using to display a carousel 167
- using to display CRM views in external applications 272
- using to display data from external application 253
- using to display fields 203

maps

- adding to mobile applications 343
- address map support 256
- API for Google maps 514
- displaying a Google map 256
- displaying Siebel CRM data in Google map 346
- jQuery plug-in 276
- Physical Renderer 276

methods

- AddMethod of the Presentation Model class 425
- AddProperty of the Presentation Model class 427
- AttachEventHandler of Presentation Model class 427
- AttachNotificationHandler of Presentation Model class 428, 432
- AttachPMBinding of Presentation Model class 428
- description of in Presentation Model 39
- for application model class 478
- for Business Component class 465
- for the JQ Grid Renderer class 476
- for the Presentation Model class 424
- guidelines for customizing the Presentation Model 97
- guidelines for using to customize the Physical Renderer 99
- Init usage 58
- Physical Renderer 56
- Presentation Model 54
- used in Presentation Model during life cycle 54
- using to add maps that include location data 343

- using to create shadow objects for applications 599
- using to display data from an external application in a view 256
- using to display data from external application in a view 253
- using to display Siebel CRM data in Google maps 348
- using to remove map buttons or icons 350
- using to render a carousel 173
- using with browser script 603

mobile

- customizing for repeating calendar events 212
- customizing to render grids 176
- server disconnected screens and views 590
- swipe and zoom support 18

O

object manager

- configuring for infinite scroll 341

P

phone

- iPhone swipe delete 276
- layout support 20
- specifying support for 282
- using to get help from Oracle 35

Physical Renderer

- description 39
- guidelines for usage with the Presentation Model 97
- guidelines for using with an client object 100
- methods of 56
- mobile renderers 275
- usage guidelines 99
- usage in Open UI 39
- using to configure scrolling 337
- using to create mobile lists 299
- using to display data from external application 253
- using to display fields 203
- using to render carousels 167
- using to render grids 176

Presentation Model

- class and method descriptions 424
- customization guidelines 97
- description 39
- example of using to render applets 47
- guidelines for customizing to render client object 100
- guidelines for usage with the Physical Renderer 99

- methods it uses 54
- usage in Open UI 39
- using to display data from external applications 253

properties

- tasks 104

R**reference**

- browser script compatibility 598

reference, APIs

- component class 465

reference, file organization

- high interactivity mode or standard interactivity mode 124

reference, internationalization support 588**S****screens**

- views, list of 591

scripting

- browser script object types 600
- creating shadow objects 599
- creating shadow objects for applets, business components or business services 599
- handling custom client script 599

search

- expression in browser script 604
- support in mobile applications 276

Siebel Tools

- preparing 104
- support for customizing 19
- using before you customize the Presentation Model 98
- using to configure menus 292
- using to customize event styles 209
- using to display calendar timestamps 213
- using to display maps 347
- using to modify look and feel 288
- using to render a grid 150, 163, 180, 182, 191, 199, 214, 218, 257, 266, 282, 285, 287, 296, 306, 312, 321, 324, 326, 328, 342, 344
- view names in 590

Siebel web templates

- folder where stored 277
- guideline for using before customizing the Presentation Model 98
- prefixing for mobile applications 277
- served from the Siebel Server 54
- usage for layout configurations 29
- usage with Google maps 515

- using to add maps 344
- using to add toggle controls 317
- using to configure height of scroll area 341
- using to customize mobile layout 280
- using to determine if Open UI is enabled for mobile 278
- using to display data from an external application 258
- using to display data in maps 348
- using to modify look and feel 288
- using to remove buttons or icons 351
- using to specify landscape or portrait layout 282
- using to toggle row visibility 320

SRF

- avoiding modifying to display list applet as a carousel 173
- customizing 19
- modifying to display data from an external application 254
- modifying to display maps 345
- usage to get shadow objects 599

T**tasks**

- properties 104

third-party

- control 50
- JavaScript plugin for Mobile 276
- JavaScript renderer 39
- Jquery FullCalendar control 48
- package 168
- resource 18
- swipeDelete event handler 343
- user interface 20
- where library must reside 123

U**URL**

- display in a field 177
- symbolic usage with external application 256

V**view**

- Affiliations 203, 254
- Calendar 211
- configuring for landscape or portrait layout 282
- Contact List 253
- displaying data from external application in 253
- example of modifying look and feel of 289
- Opportunity List 253

usage in mobile applications 590

web templates for 277

