Siebel

Configuring Siebel Open UI

October 2026

Siebel
Configuring Siebel Open UI

October 2026

F87448-08

Copyright © 1994, 2026, Oracle and/or its affiliates.

Author: Sukanya Mishra

Contents

Get Help i

1 What's New in This Release 1
What's New in Configuring Siebel Open UI, Siebel CRM 26.6 Update 1
What's New in Configuring Siebel Open UI, Siebel CRM 26.1 Update 1
What's New in Configuring Siebel Open UI, Siebel CRM 24.9 Update 1
What's New in Configuring Siebel Open UI, Siebel CRM 24.6 Update 2
What's New in Configuring Siebel Open UI, Siebel CRM 23.7 Update 2
What's New in Configuring Siebel Open UI, Siebel CRM 21.7 Update 2

2 Overview of Siebel Open UI 5
Overview of Siebel Open UI 5
About Siebel Open UI 5
How Siebel CRM Renders Siebel Open UI Clients 9
About Using This Book 12

3 Architecture of Siebel Open UI 19
Architecture of Siebel Open UI 19
About the Siebel Open UI Development Architecture 19
Life Cycle of User Interface Elements 36

4 Example of Customizing Siebel Open UI 43
Example of Customizing Siebel Open UI 43
Roadmap for Customizing Siebel Open UI 43
Process of Customizing the Presentation Model 44
Process of Customizing the Physical Renderer 63
Process of Customizing the Plug-in Wrapper 74
Configuring the Manifest for the Recycle Bin Example 85
Configuring the Manifest for the Color Box Example 86
Testing Your Modifications 87

5 Customizing Styles, Applets, Fields, and Controls 89
Customizing Styles, Applets, Fields, and Controls 89
Customizing Logos, Themes, Backgrounds, Tabs, Styles, and Fonts 89
Customizing Applets 104
Customizing Controls 170

6 Customizing Calendars and Schedulers 187
Customizing Calendars and Schedulers 187
Customizing Calendars 187
Customizing Resource Schedulers 194

7 Configuring Siebel UI to Interact with Other Applications 231
Configuring Siebel Open UI to Interact with Other Applications 231
Displaying Data from External Applications in Siebel Open UI 231
Displaying Data from Siebel Open UI in External Applications 262
Web Engine HTTP TXN Business Service 276

8 Customizing Siebel Open UI for Siebel Mobile Disconnected 289
Customizing Siebel Open UI for Siebel Mobile Disconnected 289
Overview of Customizing Siebel Open UI for Siebel Mobile Disconnected 289
Doing General Customization Tasks for Siebel Mobile Disconnected 292
Customizing Siebel Pharma for Siebel Mobile Disconnected Clients 305
Customizing Siebel Service for Siebel Mobile Disconnected Clients 315
Methods You Can Use to Customize Siebel Mobile Disconnected 324

9 Siebel Web Component Framework 351
Siebel Web Component Framework 351
Overview of the Siebel Web Component Framework 351
Architecture of the Siebel Web Component Framework 352
Execution Flow with Oracle JET Components 352
Sample JSON Structure for UI Definition 353
Customizing Using Oracle JET Components in the Web Component Framework 362
Supported Controls in Siebel Web Component Framework 385

10 Application Programming Interface 389
Application Programming Interface 389

Overview of the Siebel Open UI Client Application Programming Interface 389
Methods of the Siebel Open UI Application Programming Interface 390
Methods for Pop-Up Objects and Property Sets 468

11 Reference Information for Siebel Open UI 481

Reference Information for Siebel Open UI 481
Life Cycle Flows of User Interface Elements 481
Notifications That Siebel Open UI Supports 498
Property Sets That Siebel Open UI Supports 513
Siebel CRM Events That You Can Use to Customize Siebel Open UI 514
Languages That Siebel Open UI Supports 524
Screens and Views That Siebel Mobile Uses 527
Controls That Siebel Open UI Uses 533
Browser Script Compatibility 535

12 Post-Upgrade Configuration Tasks 541

Post-Upgrade Configuration Tasks 541
Updating Physical Renderer Customizations for Controls 541
Modifying Physical Renderer Code for Event Helper 544
Overriding Plug-In Wrappers 547

Get Help

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at *https://docs.oracle.com/*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an email to: *oracle_fusion_applications_help_ww_grp@oracle.com*.

1 What's New in This Release

What’s New in Configuring Siebel Open UI, Siebel CRM 26.6 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Siebel Web Component Framework	Modified chapter. This chapter provides an overview of Web Component Framework.

What’s New in Configuring Siebel Open UI, Siebel CRM 26.1 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Siebel Web Component Framework	New chapter. Describes how to build UI with PM driven UI rendering using JSON configuration.

What’s New in Configuring Siebel Open UI, Siebel CRM 24.9 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Configuring List and Form Applets to display a color based on the value selected	New topic. Using the Applet Presentation Model (PM) user property, you can configure a list applet column (s) or form applet control (s) to highlight a field with colors based on the value selected.

What's New in Configuring Siebel Open UI, Siebel CRM 24.6 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Customizing Applets for Homepage Views in Redwood Theme	New topic. Describes how to modify the web template to convert Homepage Views in Redwood Theme.
Customizing the Redwood Theme	New topic. Describes elements of the Redwood theme that can be customized.

What's New in Configuring Siebel Open UI, Siebel CRM 23.7 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Customizing Form Applets with Icon Map Images	New topic. Describes how to configure icon map images to customize form applets.

What's New in Configuring Siebel Open UI, Siebel CRM 21.7 Update

The following information lists the changes in this revision of the documentation to support this release of the software.

Topic	Description
Various topics.	Modified topics. As of Siebel CRM 21.2 Update, the **`applicationcontainer`** directory has been replaced by two directories, as follows: • **`applicationcontainer_external`** (for Siebel Application Interface) • **`applicationcontainer_internal`** (for all other Siebel Enterprise components)

Topic	Description
	In the Siebel Application Interface installation, web artifacts for application configurations, which were formerly located in **`applicationcontainer\webapps\siebel`**, now map to **`applicationcontainer_external\siebelwebroot`**. This directory contains subdirectories such as **`files`**, **`fonts`**, **`htmltemplates`**, **`images`**, **`migration`**, **`scripts`**, and **`smc`**. For more information, see *Siebel Installation Guide* .

2 Overview of Siebel Open UI

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*
- *How Siebel CRM Renders Siebel Open UI Clients*
- *About Using This Book*

About Siebel Open UI

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

- *Overview of Siebel Open UI*
- *Example Customizations That You Can Make with Siebel Open UI*
- *Open Development Environment*
- *Siebel Open UI JavaScript API Support*
- *Multiple Client Environment*
- *Support for More Than One Usage*
- *New Notification User Interfaces*
- *Mobile Environments*

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 Siebel CRM 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 technologies, such as HTML, CSS, or JavaScript. Siebel Open UI uses these technologies 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 CRM 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 client logic

You can modify each of these items separately and independently of each other. Siebel Open UI dynamically adjusts itself to the screen space available on the device and platform from which it is being accessed. Siebel Open UI will 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 website, 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*. For more information about the JavaScript API that Siebel Open UI uses, see *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* .

Example 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.
- Display and hide fields.
- Configure a spell checker.
- 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.
  - Display a Google map.
  - Use cascading style sheets to modify HTML elements, including position, and dimension of an element.
- Use HTML to customize the logo that your company uses or to customize the background image.
- Use JavaScript to configure menus, menu items, and the layout.
- 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.

For more information about these examples, see *Customizing Siebel Open UI*.

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. You can use a predefined, uncustomized deployment, or you can use Siebel Tools to customize Object Definition Htmls. You can use only Web development or you can use Siebel Tools and Web development depending on your 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 *Customizing Siebel Open UI*. For more information about the JavaScript API that Siebel Open UI uses, see *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 a plug-in, proprietary development environment, or native development environment that you choose, to create a custom rendering architecture that resides on top of the JavaScript API that Siebel Open UI uses.
- Use intraworkspace communication and DOM (Document Object Model) access and manipulation through JavaScript.
- Do a limited pilot test of your customizations in your current Siebel Server implementation.
- Preserve your existing customizations.

Siebel Open UI JavaScript API Support

The JavaScript API that Siebel Open UI uses is recommended over 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 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 *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. Whether a user accesses Siebel CRM using 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.
- 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 Developer Web Client (also known as the Dedicated Web Client or Thick Client)

Support for More Than One Usage

Siebel Open UI adjusts to the unique attributes that each client uses 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 user and for the expert user. An administrator can configure Siebel Open UI to meet some of these individual skill levels. Siebel Open UI can do the following:

- Support applications that you 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, use bookmarks, zoom, swipe, printing and print preview, and spell 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*.

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 Mobile Guide: Connected* .

Siebel Open UI architecture follows Responsive Web Design patterns, which allow the same content to be displayed differently based on the device from which it is being accessed.

How Siebel CRM Renders Siebel Open UI Clients

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

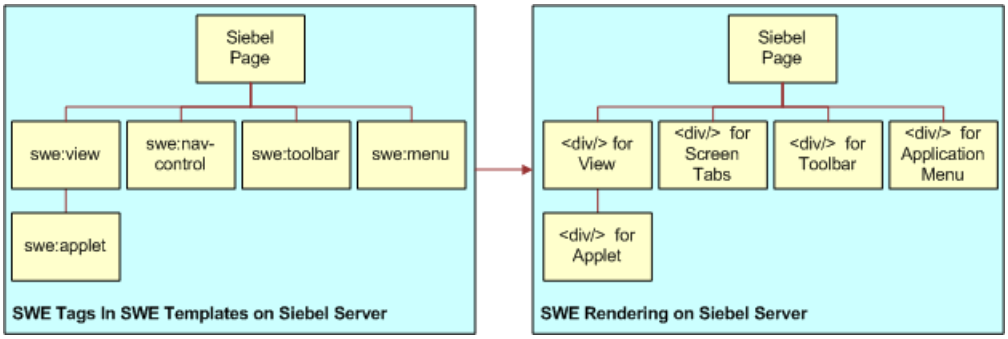
- Uses HTML div elements and HTML tables in SWE templates to determine physical layout instead of the HTML frames that high interactivity uses. Siebel Open UI does not use div elements to structure a page. The entire page hierarchy that Siebel Open UI uses is a hierarchy of div elements. Siebel Open 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.

This configuration is more closely aligned with current guidelines for Web design than the configuration that the high interactivity client usesd. 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 ODH or create a new ODH.

How Siebel CRM Renders Div Containers on Siebel Servers

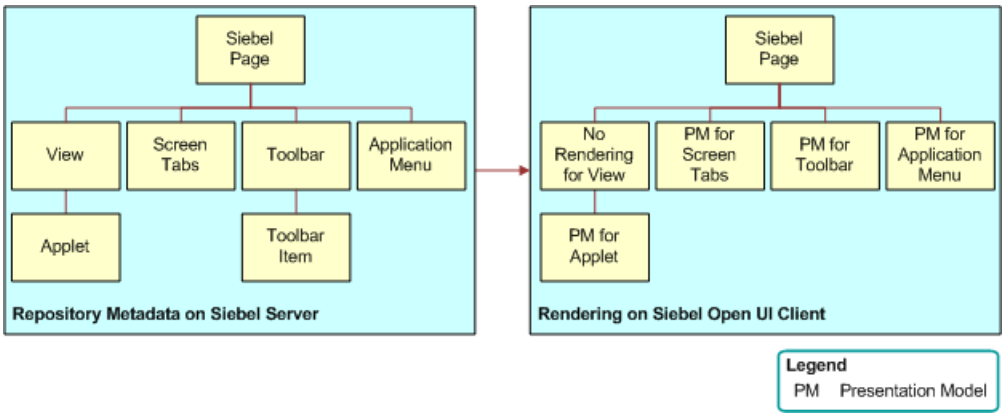
The following figure illustrates how the Siebel Server uses OD tags that reside in ODH to render div containers on the Siebel Server. For example, it renders:

- A tag with type od-type="view" as a View container.
- A tag with type od-type="nav-control" as a Screen Tab container.
- A tag with type od-type="toolbar" as a Toolbar container.
- A tag with type od-type="menu" as an Application Menu container.
- A tag with type od-type="applet" as an Applet container.



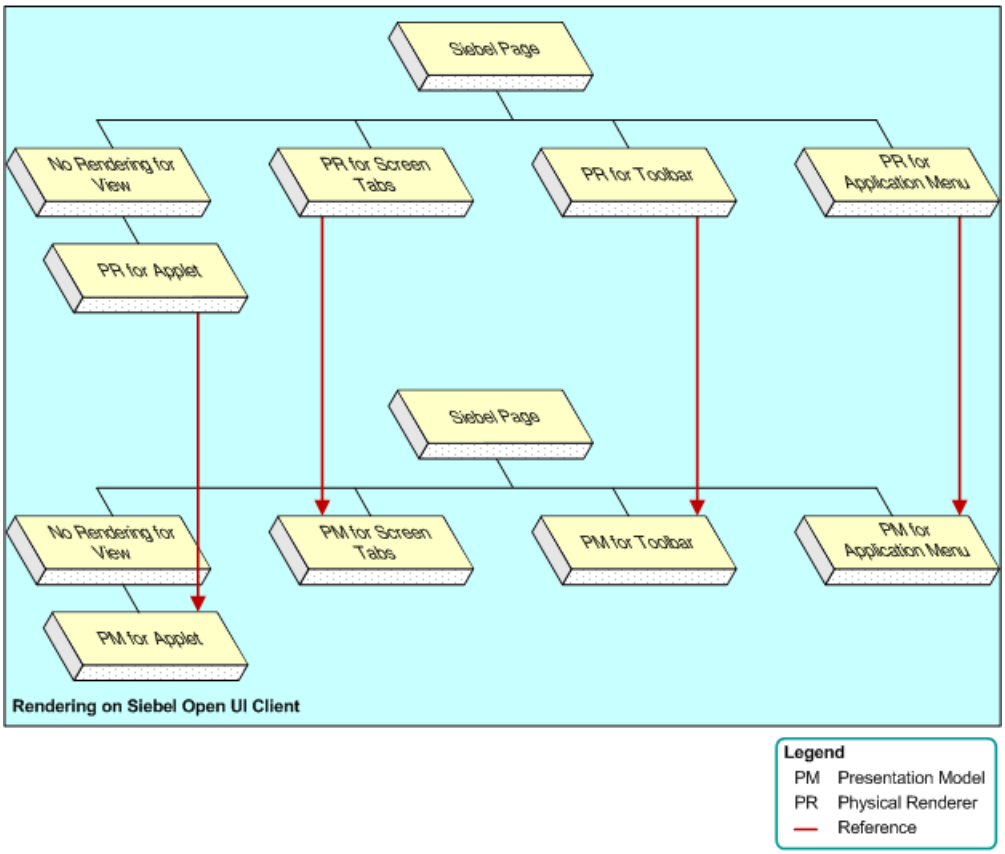
How Siebel CRM Handles Data in Siebel Open UI

The following figure 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.



How Siebel CRM Renders Objects in Siebel Open UI

The following figure illustrates how Siebel CRM uses a physical renderer, which is a JavaScript file that Siebel Open UI uses to render 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*.



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 so that it uses 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 Carousels*.

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 *Example Customizations That You Can Make with 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. You can customize at the control level writing plug-in wrappers that govern how a control should appear and behave when a certain set of conditions are satisfied. A check box appearing as a flipswitch on mobile devices is an example of this type of implementation.

About Using This Book

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

- *Important Terms and Concepts*
- *How This Book Indicates Computer Code and Variables*
- *How This Book Describes Objects*
- *About Siebel CRM Releases*
- *Support for Customizing Siebel Open UI*
- *Getting Help from Oracle*

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 Runtime Repository. For example, a screen, view, applet, business component, menu, or control is each an example of a Siebel object. The Contact List Applet is an example of a Siebel CRM applet. 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 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 or 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. For more information, see *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.
- 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 the glossary.
- Using the Siebel Open UI client, see *Siebel Fundamentals* for Open UI.
- Using Siebel Tools, see *Using Siebel Tools* .

How This Book Indicates Computer Code and Variables

Computer font indicates a value that 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 syntax 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:

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

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

```
function Recyclebinpmodel(){
 SiebelAppFacade.RecycleBinPModel.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, in some instances this book describes only 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 that the applet contains, 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, see *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 Siebel CRM Releases

Before you can perform the configuration tasks described in this book, you must install Siebel CRM and perform postinstallation tasks. For more information, see *Siebel Installation Guide* .

Depending on the software configuration that you purchase, your Siebel CRM products 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. You configure the same Object Definition 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 *Application Programming Interface*. Oracle continues to support browser scripting in previous releases, 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 CRM 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 CRM supports integrating external content in the Siebel Open UI client.
- You can modify the cascading style sheets that come predefined with Siebel Open UI to rebrand your deployment and customize the user experience.
- Siebel Open UI supports usage of Siebel SmartScript to specify workflow. For more information, see *Siebel SmartScript Administration Guide* .
- 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 library, 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.
- 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*.

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 *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 that you make. For more information, see *About the Presentation Model* and *About the Physical Renderer*.

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

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*
- *Life Cycle of User Interface Elements*

About the Siebel Open UI Development Architecture

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

- *Overview of the Siebel Open UI Development Architecture*
- *Example of How Siebel Open UI Renders a View or Applet*
- *Customizing the Presentation Model and Physical Renderer*
- *Customizing the Physical Renderer*
- *Customizing a Plug-in Wrapper*
- *Stack That Siebel Open UI Uses to Render Objects*
- *Items in the Development Architecture You Can Modify*
- *Example Client Customizations*
- *Differences in the Server Architecture Between High Interactivity and Siebel Open UI*
- *Differences in the Client Architecture Between High Interactivity and Siebel Open UI*

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. 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:

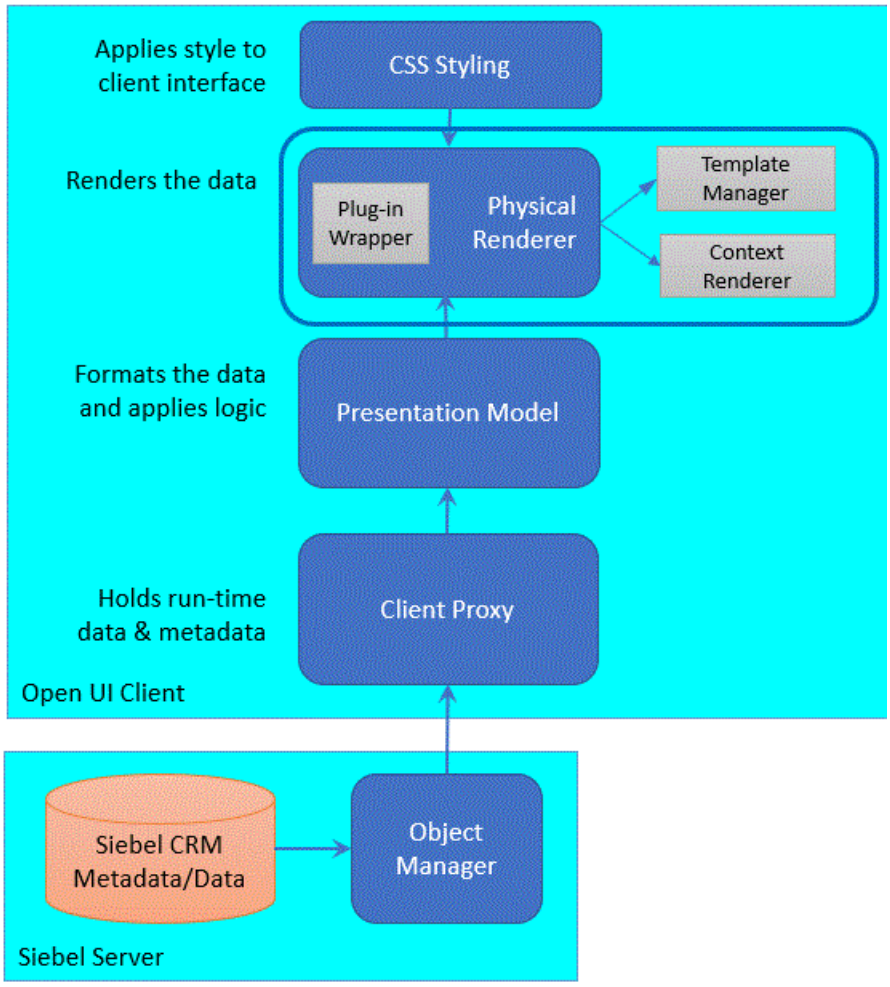
- Application
- Screen
- View
- Applet
- Applet Control
- Menu
  - Application menu
  - Applet menu
- Toolbar

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

Architecture You Can Use to Customize Siebel Open UI

The following figure illustrates the basic architecture that you can use to customize Siebel Open UI and it contains the following key components:

1. On the Siebel Open UI Client:
    - CSS Styling applies style to the client interface.
    - The Physical Renderer renders the data (using the Template Manager, Plug-in Wrapper and Builder, and Context Renderer).
    - The Presentation Model formats the data and applies the logic.
    - The Client Proxy holds runtime data and metadata.
2. The Siebel Server holds the Object Manager and Siebel CRM Metadata/Data.



For an overview of how Siebel Open UI uses the presentation model and physical renderer, see *How Siebel CRM Renders Div Containers on Siebel Servers*.

About the Presentation Model

The presentation model is a JavaScript file that specifies how to handle the metadata and data thatSiebel 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*.

About the Physical Renderer

A physical renderer is a JavaScript file that Siebel Open UI uses to render the user interface. It binds a presentation model to a 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 allows you to use custom or third-party JavaScript code to render the user interface. 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*.

About the Template Manager

The *template manager* is a JavaScript object that provides HTML markup as requested by a physical renderer, a plug-in wrapper or any other active JavaScript object running in Siebel Open UI. A template manager ensures that each component of Siebel Open UI generates exactly the same markup, enhanced with a predefined classname, for similar type of UI controls that is independent of device, browser, and resolution. For example, if a text field is being rendered in Siebel Open UI, then it must use the same classname, for example, siebui-input, whether it is being rendered in a browser on a desktop, or a mobile device.

About the Template Manager in Responsive Web Design

One of the most crucial aspects of responsive Web design is to have clean and virtually identical DOM elements within a specific classname for a control. For example, an anchor can also be styled in such a way that it appears similarly to a button in one context and in another might appear as a hyperlink.

You must, however, provide the same DOM element for a particular type consistently, coupled with a specialized classname, when required. The template manager then acts as an HTML content provider for all types of primitives controls.

How It Works

The template manager expects the caller, which in most cases would be renderers or plug-in wrappers, to provide certain information on what kind of control they need. For example, does the caller need to create input element? Depending on the type and other parameters specified by the caller, the template manager determines the control that is required, then builds an HTML string and returns that string to the caller. The template manager also provides the flexibility to add more DOM attributes which may or may not be standard, for example mobile specific "data-" attributes, or automation attributes.

For more information about the template manager class, see *Template Manager Class*.

About Event Helper Objects

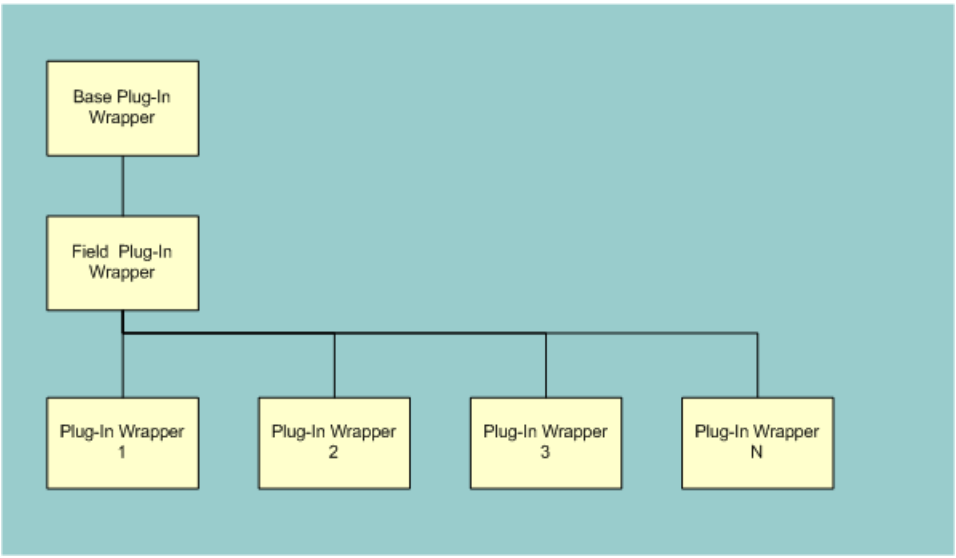
Event helper objects facilitate event binding in a physical renderer or a plug-in wrapper. They consolidate events across platforms, most importantly standardize events such as touch and click. The differences required in rendered markup and the behavioral aspects, if any, can be handled internally by the template manager and the even helper object respectively.

For more information about the event helper objects, see *About Event Helper Objects*.

About Plug-in Wrappers

A plug-in wrapper is a complete and independent manager of an applet control and its life-cycle. It is entirely responsible for all actions of a control, including but not limited to its showing, value management, event handling. Plug-in wrappers cater to control level management. A plug-in wrapper allows the wrapper to handle the control of specific functionalities. Individual renderers will delegate the control-specific-functionalities to the wrappers. The wrappers handle the applet control level implementation.

The following figure outlines the class structure of plug-in wrappers.



This figure contains the following elements:

1. **Base Plug-In Wrapper.** This is the base specification class. It defines the base properties and methods to which every plug-in wrapper must adhere. No functionality is implemented in this class and it is not recommended that any derivation or customization occur from this class.
2. **Field Plug-In Wrapper.** This is the class that defines the default functionality of a control. All APIs have a definition, and this plug-in wrapper is a fallback class for all customizations. You may choose to derive a custom wrapper from this class if your intention is to write a new customization.
3. **Plug-In Wrapper 1, Plug-In Wrapper 2, Plug-In Wrapper 3, Plug-In Wrapper N.** These are Siebel Open UI out-of-the-box customizations that are used to display specific types of controls. Examples of these are date pickers, drop-down menus, flip switches and signatures. You may choose to derive a custom wrapper from one of these classes if your intention is to slightly modify the functionality of an existing plug-in wrapper.

For more information about plug-in wrappers, including detailed instructions about creating and customizing a plug-in wrapper, see *Process of Customizing the Plug-in Wrapper* and *Plug-in Wrapper Class*.

About the Plug-in Builder

The plug-in builder wires the physical renderer to a plug-in wrapper for a given control and a given set of conditions. It also provides a decoupling between physical renderers, such as an applet, and plug-in wrappers for controls in that applet.

For more information see, *About Plug-in Wrappers* and *Plugin Builder Class*.

About Context Renderers

A context renderer is a JavaScript object that Siebel Open UI optionally uses to enhance the user interface for generic functionality. It is an extension to the physical renderer and provides loose coupling for specific UI operations. A physical renderer can have one or many context renderers. In an ideal scenario, a context renderer would always be limited to user interface operation.

For example, if all Applets in a view needs to be rendered as Accordion Panel, then it can be implemented as context renderer. Also, the context renderer can be attached to view physical renderers without any impact on the View PR functionality.

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

Siebel Open UI uses presentation models and physical renderers to bind data to the user interface.

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

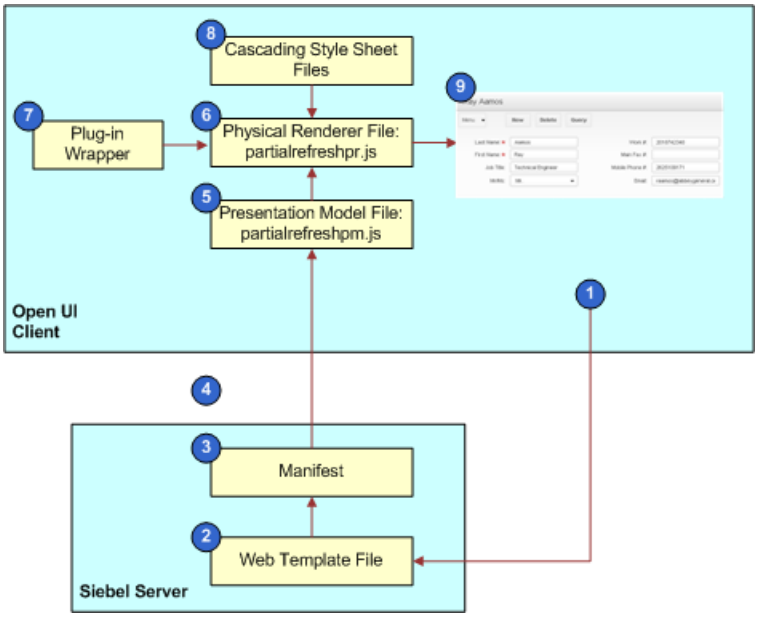
- **Physical presentation and interaction for a user interface element.** For example, a grid, carousel, form, tree, tab, menu, button, and so on.
- **Logical presentation and interaction that Siebel Open UI can physically display in more than one way.** For example, Siebel Open UI can display a list 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.

Siebel Open UI 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 an application so that a field is required or uses a hierarchical picklist. It also allows you to do the following customizations:

- **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. You can also configure Siebel Open UI to display a check box control in a grid or a form as a flip switch.

Example of How Siebel Open UI Renders a View or Applet

The following figure illustrates how Siebel Open UI renders the Contact Form Applet.

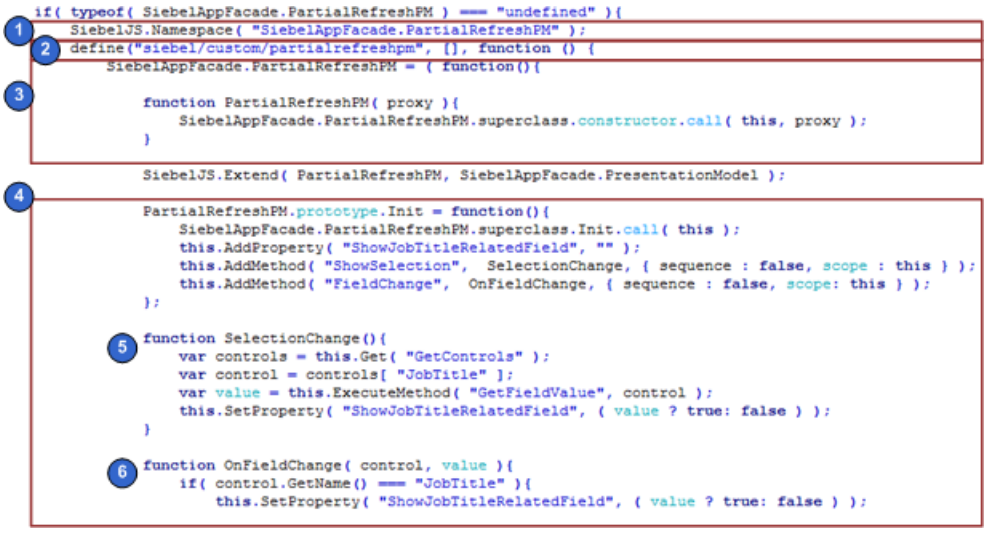


As shown in this figure, Siebel Open UI does the following to render the Contact Form Applet:

1. The user attempts to navigate to the Contact Form Applet.
2. Siebel Open UI creates the view that displays this applet.
3. Siebel Open UI references the manifest to identify the files it must download to the client. For more information, see *Configuring Manifests*.
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 *About the Presentation Model*.
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*.
7. A physical renderer fetches and incorporates plug-in wrappers for its applet controls. For more information, see *Example of a Plug-in Wrapper*.
8. One or many context renderers register themselves with a corresponding object. For more information, see *Example of a Context Renderer*.
9. Siebel Open UI loads the cascading style sheets according to the manifest configuration that it referenced in Step 3.
10. Siebel Open UI uses a presentation model, physical renderer, context renderer (optional), and cascading style sheets to render the Contact Form Applet.

Example of a Presentation_Model

The following figure 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 *Displaying and Hiding Fields*.



As shown in this figure, 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*.
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

The following figure 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 *Displaying and Hiding Fields*.

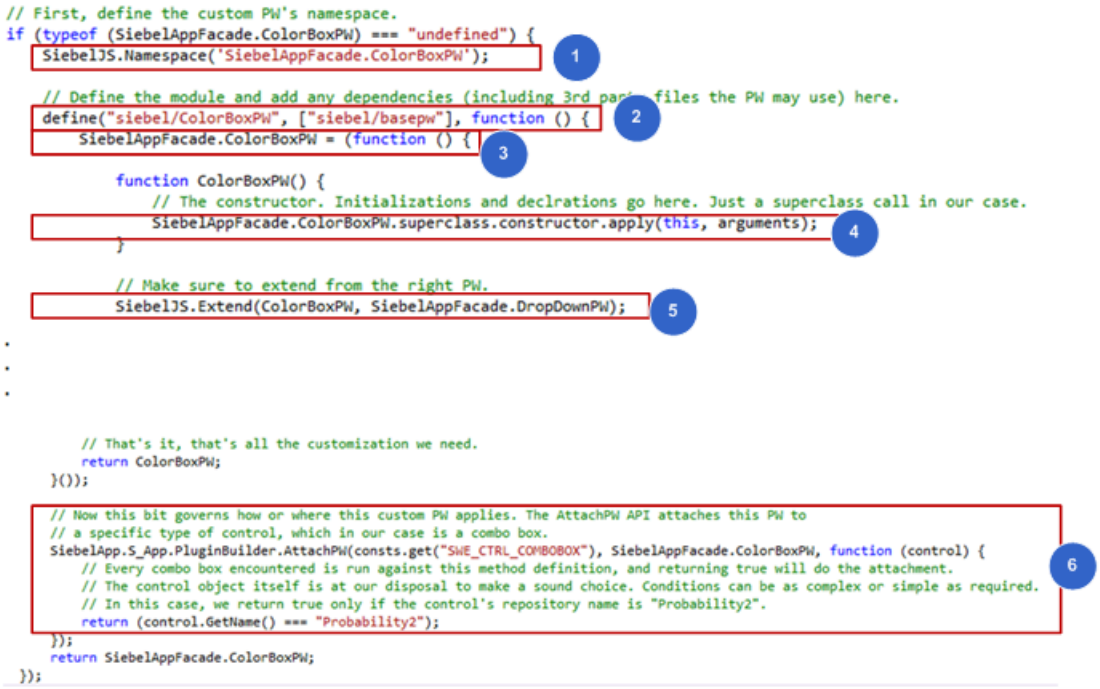


As shown in this figure, the partialrefreshpr.js file includes the following sections:

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*.
3. Creates the physical renderer class.
4. Specifies the ShowJobTitleRelatedField property.
5. Includes the logic that Siebel Open UI runs if it modifies ShowJobTitleRelatedField.

Example of a Plug-in_Wrapper

The following figure describes how the ColorBoxPW.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 *Process of Customizing the Plug-in Wrapper*.

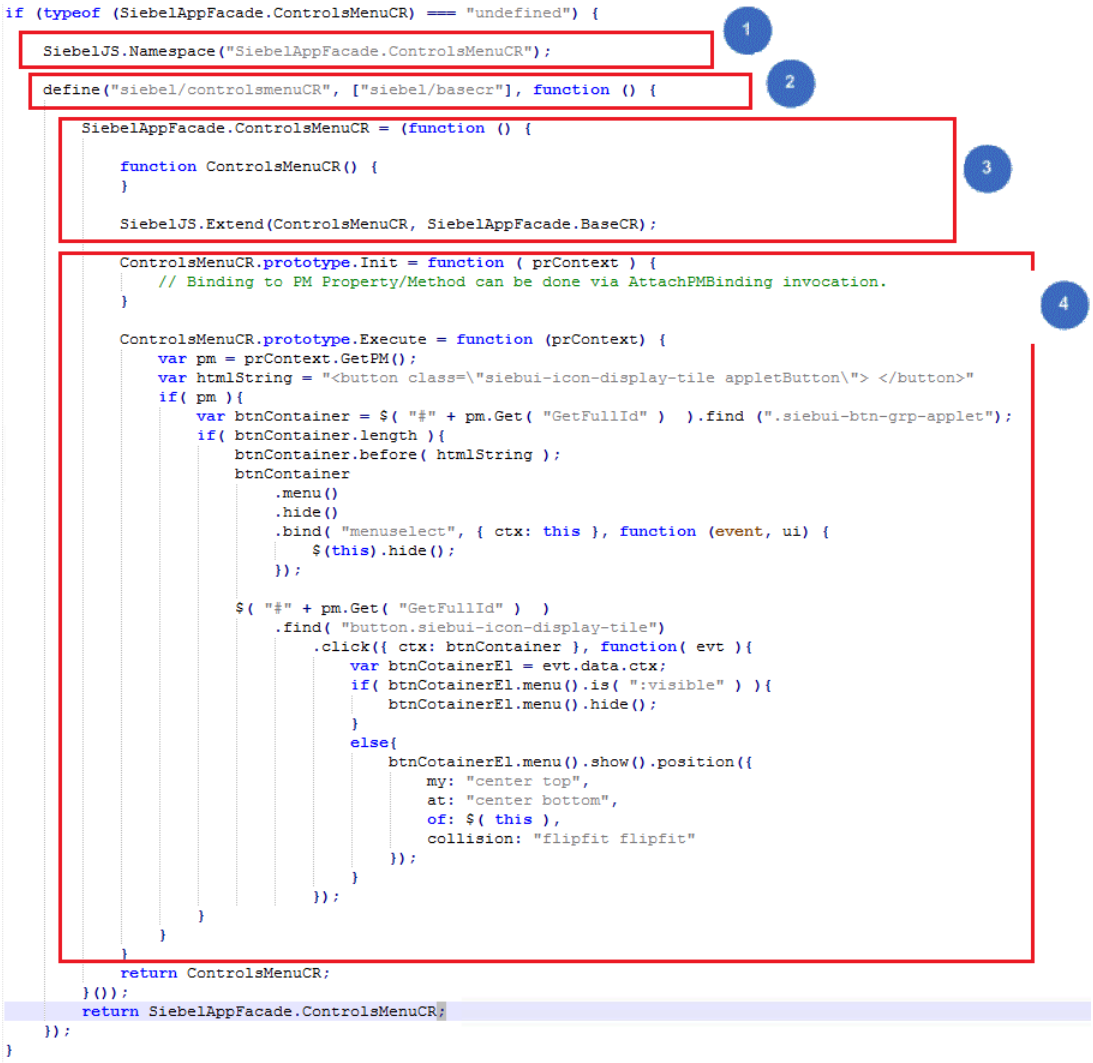


As shown in this image, the ColorBoxPW.js file includes the following sections:

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*.
3. Creates the plug-in wrapper class.
4. Implements the Life Cycle and Interface Methods of a Plug-in Wrapper.
5. Implements events handlers and other methods specific to the given Plug-in Wrapper.
6. Wires the Plug-in Wrapper to the Physical Renderer (optionally) based on conditionals.

Example of a Context Renderer

The following figure describes how the controlsmenuCR.js file does the UI transformation of button controls in a menu. When configured as a context renderer for Contact Form Applet via Manifest, it displays the Applet Button Controls in Menu instead of displaying them inline.




```
if (typeof (SiebelAppFacade.ControlsMenuCR) === "undefined") {
    SiebelJS.Namespace("SiebelAppFacade.ControlsMenuCR");
    define("siebel/controlsmenuCR", ["siebel/basecr"], function () {
        SiebelAppFacade.ControlsMenuCR = (function () {

            function ControlsMenuCR() {
            }

            SiebelJS.Extend(ControlsMenuCR, SiebelAppFacade.BaseCR);

            ControlsMenuCR.prototype.Init = function ( prContext ) {
                // Binding to PM Property/Method can be done via AttachPMBinding invocation.
            }

            ControlsMenuCR.prototype.Execute = function (prContext) {
                var pm = prContext.GetPM();
                var htmlString = "<button class=\"siebui-icon-display-tile appletButton\"> </button>"
                if( pm ){
                    var btnContainer = $( "#" + pm.Get( "GetFullId" )  ).find (".siebui-btn-grp-applet");
                    if( btnContainer.length ){
                        btnContainer.before( htmlString );
                        btnContainer
                            .menu()
                            .hide()
                            .bind( "menuselect", { ctx: this }, function (event, ui) {
                                $(this).hide();
                            });

                        $( "#" + pm.Get( "GetFullId" )  )
                            .find( "button.siebui-icon-display-tile")
                                .click({ ctx: btnContainer }, function( evt ){
                                    var btnCotainerEl = evt.data.ctx;
                                    if( btnCotainerEl.menu().is( ":visible" ) ){
                                        btnCotainerEl.menu().hide();
                                    }
                                    else{
                                        btnCotainerEl.menu().show().position({
                                            my: "center top",
                                            at: "center bottom",
                                            of: $( this ),
                                            collision: "flipfit flipfit"
                                        });
                                    }
                                });
                    }
                }
            }
            return ControlsMenuCR;
        }());
        return SiebelAppFacade.ControlsMenuCR;
    });
}
```

As shown in this image, the controlsmenuCR.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*.
3. Creates the context renderer class.
4. Implements the Life Cycle and Interface Methods of a context renderer.

Customizing the Presentation Model and Physical Renderer

Siebel Open UI uses two JavaScript files to implement the presentation model and the physical renderer and plug-in wrappers 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

It uses the following concatenated file for all applet controls:

- pwinfra.js is a concatenation of all the plug-in wrapper objects used for all standard applet controls in the Siebel application.

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 configure it to 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*.

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 Maps*.

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 defines 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 *Example of Customizing Siebel Open UI*

Customizing the Physical Renderer

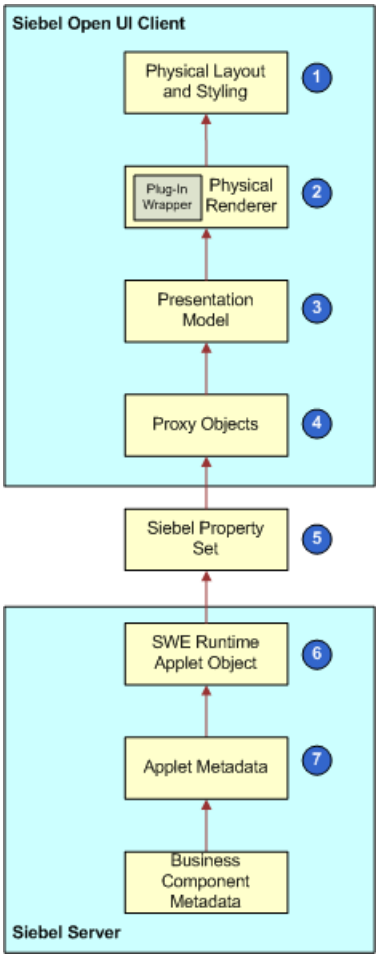
You can use a physical renderer to modify how Siebel Open UI renders an object. For example, Siebel Open UI displays the predefined Contact Affiliations list applet as a typical Siebel CRM list. You can modify this list to display as a carousel. You can modify how the user scrolls through a set of records, 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 Carousels*. For more information, see *About the Physical Renderer*.

Customizing a Plug-in Wrapper

You can use a plug-in wrapper to modify how Siebel Open UI renders an Applet Control object. For example, Siebel Open UI displays all fields with boolean values as Check Boxes. You can modify this to display them as flip switch controls. You can modify how the user sets and resets the value of the boolean field, which is a physical aspect of the applet control that a plugin wrapper defines. But this control is still a boolean field: the logical representation of the applet control that the presentation model defines. You do not modify this logical representation. To view an example of this type of modification, *Customizing a Plug-in Wrapper*. For more information, see *About Plug-in Wrappers*.

Stack That Siebel Open UI Uses to Render Objects

The following figure describes the stack that Siebel Open UI uses to render objects. It uses the applet object as an example.



As shown in this figure, the stack that Siebel Open UI uses to render objects includes the following:

1. **Physical layout and styling.** Allows you to use HTML to display content, JavaScript to customize logic, and cascading style sheets to customize layout and styling in the client. You can position or hide controls to achieve almost any layout requirement.
2. **Physical renderer.** For more information, see *About the Physical Renderer* and *About Plug-in Wrappers*.
3. **Presentation model.** For more information, see *About the Presentation Model*.
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 customize only the information that flows 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*.
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.
6. **SWE runtime 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

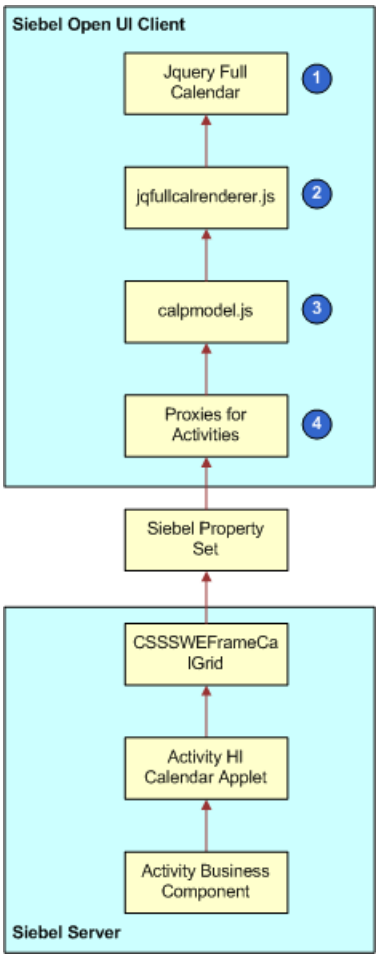
request to a business component or business service. Siebel Open UI does not currently include a scripting interface that allows you to modify the property set that the applet sends to the client.

7. **Applet metadata.** The applet object in the Siebel Runtime Repository 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 configure Siebel Open UI to get data through a presentation model in the client to develop functionality that is similar to the functionality that a more complex binding provides. For more information, see *About Objects and Metadata*.

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 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.

The following figure describes an example stack that Siebel Open UI uses to display a calendar applet.

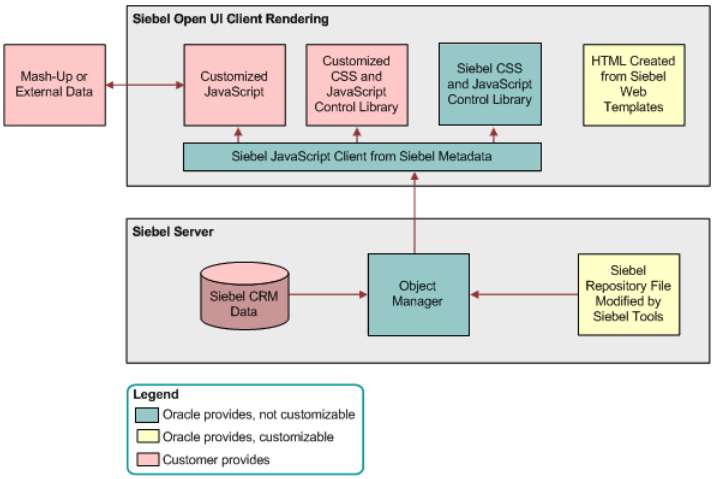


As shown in this figure, Siebel Open UI uses the following to display a calendar applet:

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 a list applet.
4. **Activity proxies.** Includes proxies for the Activity Calendar Applet and the Activity business component.

Items in the Development Architecture You Can Modify

The following figure indicates the predefined items in the development architecture that Siebel CRM provides and the items that you can modify. It delineates areas where you can customize Siebel Open UI.



As shown in this figure:

1. The Siebel Open UI Client Rendering side contains the following:
    - Oracle provided, non-customizable components: Siebel JavaScript Client from Siebel Metadata, Siebel CSS and JavaScript Control Library.
    - Oracle provided, customizable components: HTML Created from Web Templates.
    - Customer provided components: Customized JavaScript, Customized CSS and JavaScript Control Library, Mash-up of External Data.
2. The Siebel Server side contains the following:
    - Oracle provided, non-customizable components: Object Manager.
    - Oracle provided, customizable components: Siebel Repository File (modified by Siebel Tools/Web Tools.
    - Customer provided components: Siebel CRM Data.

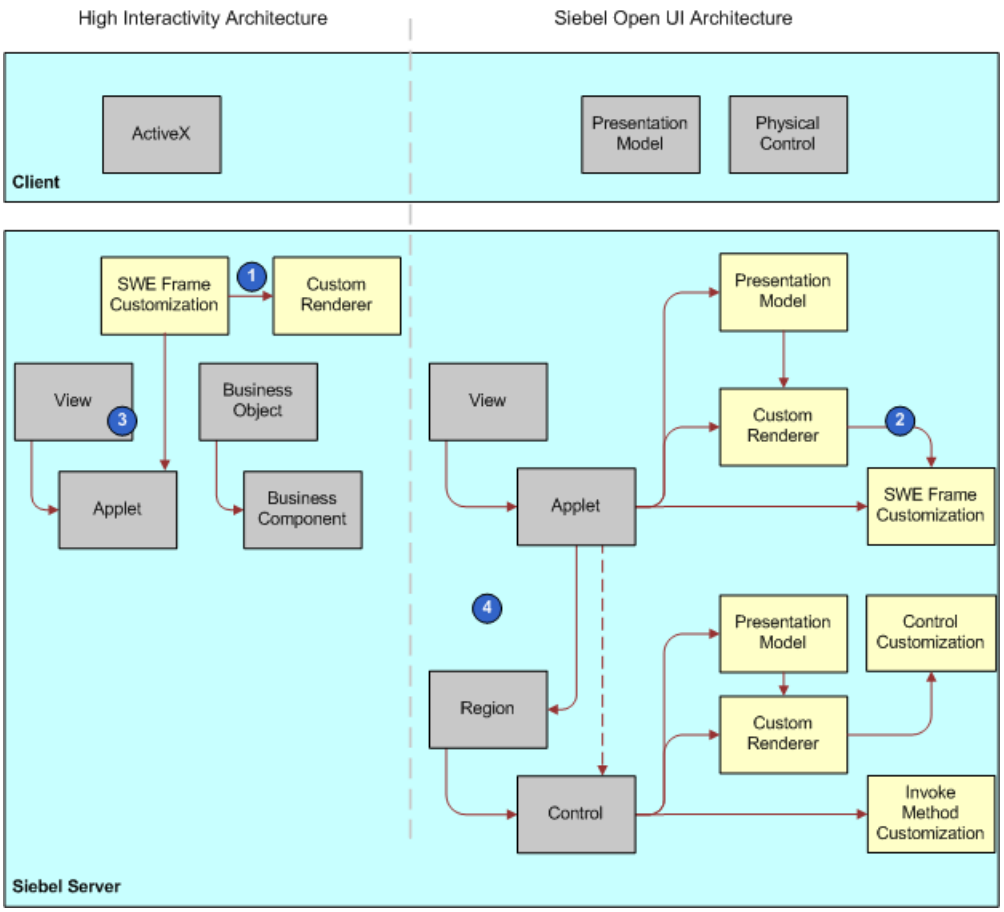
Example Client Customizations

The following table describes some example client customizations you can do in Siebel Open UI. For detailed examples, see *Customizing Siebel Open UI*.

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 *Using Siebel Tools* .
Add custom client behavior.	You modify a presentation model. For example: **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. 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. **Modify the theme of a page.** For example, you can configure Siebel Open UI to modify the theme of a page if the user changes the orientation of a tablet device. You 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 control to render the presentation model. For example, to render a list applet as a carousel, you use the appropriate third-party control.
Add specific applet control-level behavior and rendering.	For example, you can customize plug-in wrappers to make a boolean field render and behave like a flip switch, rather than a check box.
Position controls and customize style.	You can modify CSS files.

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

The following figure compares the server architecture between the old high interactivity client and Siebel Open UI.

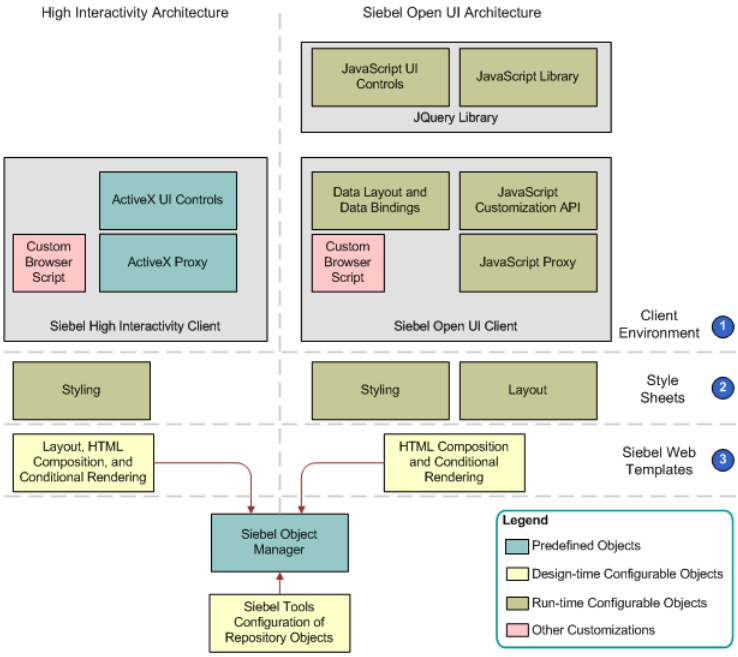


As shown in this figure, the things to note when comparing the Server architecture that high interactivity uses and the Server architecture that Siebel Open UI uses include the following:

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

The following figure compares the ActiveX UI architecture that the old high interactivity client used to the architecture that Siebel Open UI uses.



As shown in this figure, things to note when comparing the client architecture between high interactivity and Siebel Open UI include the following:

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 Application Interface serves static style sheets.
3. **Object Definition Html.** The Siebel application serves dynamic Object Definition Htmls.

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 call 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*.
- Process 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 *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*:

- **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 *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*. For more information, see *Get Method*.
- **AddMethod.** Adds a method to the presentation model. For more information, see *AddMethod Method*.
- **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* and *AttachEventHandler Method*.
- **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*.
- **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*.
- **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:

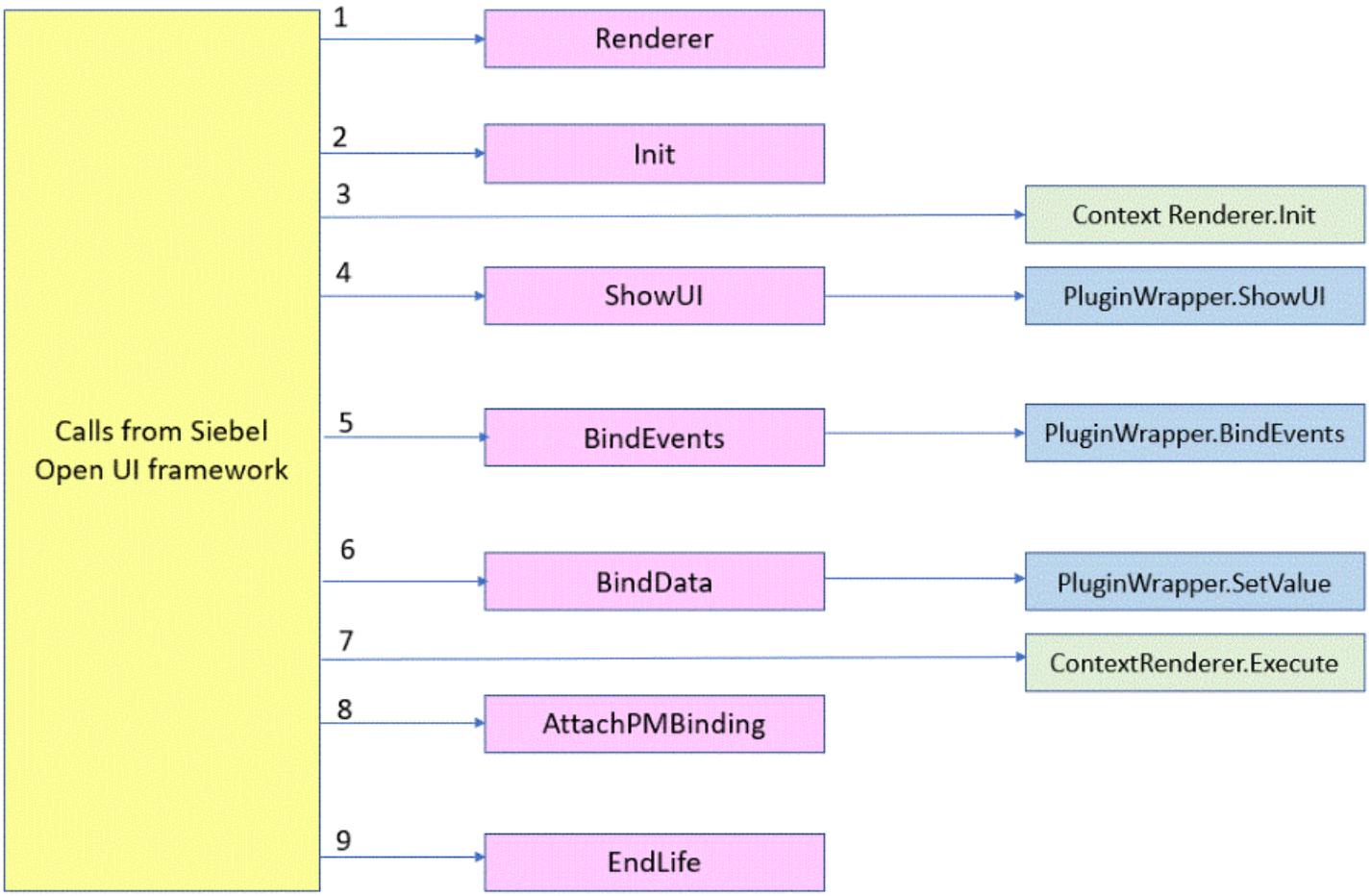
- **Get.** Gets the value of a property that resides in a presentation model.
- **ExecuteMethod.** Runs a method that the AddMethod method calls. For more information, see *ExecuteMethod Method*.
- **OnControlEvent.** Calls an event. The physical renderer uses the OnControlEvent method to call the presentation model and send an event. To call the method, the presentation model uses a binding that exists between the event and the presentation model method and the AttachEventHandler method. For more information, see *OnControlEvent Method* and *AttachEventHandler Method*.
- **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*.

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*. For an example that uses Setup, see *Customizing the Setup Logic of the Presentation Model*.

Life Cycle of a Physical Renderer

The following figure illustrates the life cycle of a physical renderer. For examples of various life cycle flows, see *Life Cycle Flows of User Interface Elements*.



As shown in this figure, 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*.
3. The Open UI framework makes a call to the context renderer's Init function after instantiating it. Siebel Open UI adds the corresponding physical renderer's instance to this function and the implementation can use the physical renderer or presentation model's interface.

4. **ShowUI.** Displays a physical control that corresponds to an applet control. It renders the container for the metadata, data, and 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. Also, ShowUI calls all of the plug-in wrappers of the associated applet controls. For more information, see *ShowUI Method*.
5. **BindEvents.** Sets up the user interface binding of 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. Also, BindEvents calls all of the plug-in wrappers of the associated applet controls. For more information, see *BindEvents Method*.
6. **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. Also, BindData calls all of the plug-in wrappers of the associated applet controls. For more information, see *BindData Method*.
7. The context renderer's Execute method is called by the framework immediately after the physical renderer's lifeCycle method execution, that is, after the ShowUI, BindData, BindEvents functions. OpenUI also adds the corresponding physical renderer's instance to this function and the implementation can use the physical renderer or presentation model's interface.
8. **AttachPMBinding.** Attaches handlers to notifications that occur during the life cycle. For more information, see *AttachPMBinding Method*. For more information about notifications that can occur during the life cycle, see *Notifications That Siebel Open UI Supports*.

   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* and *OnControlEvent Method*.
9. **EndLife.** Ends the life of the physical renderer. For more information, see *EndLife Method*.

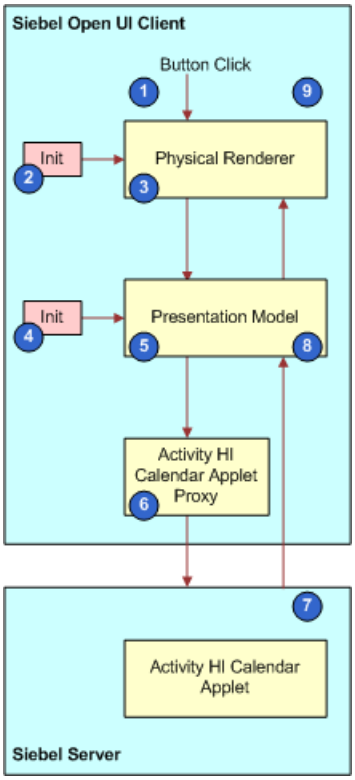
Life Cycle of a Plug-in Wrapper

The plug-in wrapper uses methods in the following sequence:

1. **ShowUI.** Performs show related activities for a control. For more information see *ShowUI Method*.
2. **BindEvents.** Attaches events to the DOM instance of the control. For more information see *BindEvents Method*.
3. **BindData.** Initializes data to the DOM instance of the control. For more information see *BindData Method*.
4. **EndLife.** Ends the life of the Plug-in Wrapper. For more information see *EndLife Method*.

Example of the Life Cycle of a User Interface Element

The following figure describes the life cycle of the calendar user interface element.



As shown in this figure, the following sequence of events 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:

   ```
   OnControlEvent("Refresh_Calendar",RequestCalendarData)
   ```

   For more information, see *OnControlEvent Method*.
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 ("GetCalendarOUIData", ProcessCalendarData)
   AttachPostProxyExecute ("GetCalendarOUIData",SetDefaultFocus)
   ```

   For more information, see *AttachEventHandler Method*.
5. The presentation model sends the RequestCalendarData method to the Activity Calendar Applet proxy.
6. The Activity Calendar Applet proxy sends a request to the Siebel Server to call the RequestCalendarData method.
7. The Siebel Server gets metadata from the Activity 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*.
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

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*
- *Process of Customizing the Presentation Model*
- *Process of Customizing the Physical Renderer*
- *Process of Customizing the Plug-in Wrapper*
- *Configuring the Manifest for the Recycle Bin Example*
- *Configuring the Manifest for the Color Box Example*
- *Testing Your Modifications*

Roadmap for Customizing Siebel Open UI

You do the following tasks to customize Siebel Open UI:

- *Process of Customizing the Presentation Model*
- *Process of Customizing the Physical Renderer*
- *Process of Customizing the Plug-in Wrapper*
- *Configuring the Manifest for the Recycle Bin Example*
- *Configuring the Manifest for the Color Box Example*
- *Testing Your Modifications*

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 the 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 that 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.

- **Modify a plug-in wrapper.** You customize a specific control by writing a plug-in wrapper (PW). In this example, if the customization is on the Opportunity List applet, a custom PW will be written for the probability field which will add a colorbox to the field, which will then change colors based on the value in the probability field. Also, clicking on the box will open a legend that explains the colors.

For background information about the architecture that this example uses, see *Stack That Siebel Open UI Uses to Render Objects* and *Life Cycle of User Interface Elements*.

Process of Customizing the Presentation Model

This task is a step in *Roadmap for Customizing Siebel Open UI*.

To customize the presentation model, do the following tasks:

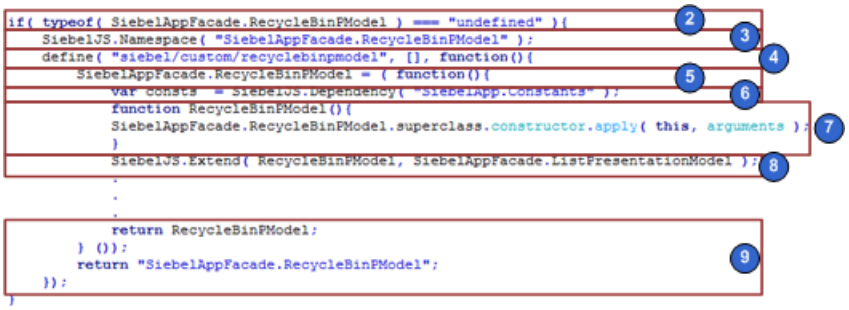
1. *Creating the Presentation Model*
2. *Customizing the Setup Logic of the Presentation Model*
3. *Customizing the Presentation Model to Identify the Records to Delete*
4. *Customizing the Presentation Model to Delete Records*
5. *Overriding Predefined Methods in Presentation Models*
6. *Customizing the Presentation Model to Handle Notifications*
7. *Attaching an Event Handler to a Presentation Model*
8. *Customizing Methods in the Presentation Model to Store Field Values*
9. *Customizing the Presentation Model to Call the Siebel Server and Delete a Record*

Creating the Presentation Model

This task is a step in *Process of Customizing the Presentation Model*.

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*.

The following figure 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.



To create the presentation model

1. Create the custom presentation model file:

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

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\custom
```

This topic describes how to modify code that resides in the recyclebinpmodel.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*. For more information about the language_code, see *Languages That Siebel Open UI Supports*.

b. Use a JavaScript editor to open the recyclebinpmodel.js file that you downloaded in Step a.

2. Make sure 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 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/recyclebinpmodel", [], function(){
```

You must use the Define method to make sure 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*.

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:

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

For more information about injected dependency, see *About Dependency Injection*.

9. Return the constructor:

```
return RecycleBinPModel;
 } ());
 return "SiebelAppFacade.RecycleBinPModel";
});
```

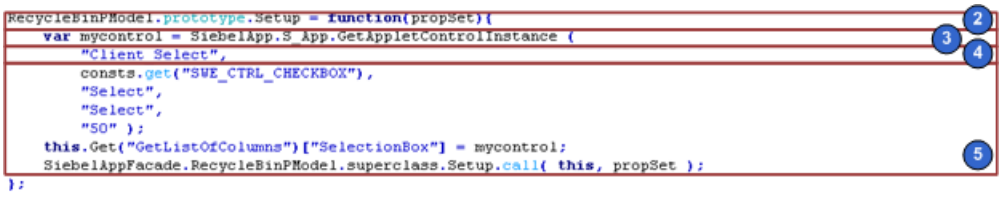
10. Save the recyclebinpmodel.js file.

Customizing the Setup Logic of the Presentation Model

This task is a step in *Process of Customizing the Presentation Model*.

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.

The following figure 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.



To customize the setup logic of the presentation model

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

   To do this identification, you can examine the JavaScript API methods to identify the method that most closely matches the behavior that your example requires. For more information about this JavaScript API, see *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*.
   - **Applet methods.** For more information, see *Presentation Model Class for Applets*.
   - **List applet methods.** For more information, see *Presentation Model Class for List Applets*.
   - **Applet control methods.** For more information, see *Applet Control Class*.
   - **Menu methods.** For more information, see *Presentation Model Class for Menus*.
   - **Siebel business service methods.** For more information, see *Business Service Class*.

   In this example, you can examine the presentation model that Siebel Open UI uses for list applets to identify the property or method that the object you must modify uses. To identify this property, see *Properties of the Presentation Model That Siebel Open UI Uses for Applets*.

   After examining these properties, assume that you determine that Siebel Open UI uses the GetListOfColumns method that the presentation model references. 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:

   ```
   RecycleBinPModel.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*.

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 to 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*.

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:

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

where:

- `consts.get("SWE_CTRL_CHECKBOX")` specifies the control as a check box.
- `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*.

6. Save the recyclebinpmodel.js file.

Customizing the Presentation Model to Identify the Records to Delete

This task is a step in *Process of Customizing the Presentation Model*.

In this topic, you modify the list column control that you created in Step 3. This control uses a check box, so you must make sure that Siebel Open UI stores the value of this check box when the user toggles it.

The following figure 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 );                    1
    this.AddMethod( "LeaveField", PreLeaveField, { sequence : true, scope : this } );
    this.AddProperty( "DeletionPendingSet",     [] );                                 6
    .
    .
    .
function PreLeaveField( control, value, notLeave, returnStructure ){
    if (control.GetName() === "Client Select"){                                       2
        this.ExecuteMethod( "SetActiveControl", null );                               5
        var delObj = this.Get( "DeletionPendingSet" );
        var currentSelection = this.Get( "GetSelection" );
        if( value === "Y" ){
            delObj[currentSelection] = this.Get("GetRecordSet")[currentSelection];
        }
        else{
            delObj[currentSelection] = null;
        }                                                                             7
        returnStructure[ "CancelOperation" ] = true;                                  3
        returnStructure[ "ReturnValue" ] = true;                                      4
    }
}
```

To customize the presentation model to identify the records to delete

1. In the recyclebinpmodel.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 that you must add when you do your own customization work, you can examine the life cycles that Siebel Open UI uses that most closely meets your business requirement. To view these life cycles, see *Life Cycle Flows of User Interface Elements*.
   - 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* and *Flow That Handles Focus Changes in List Applets*.
   - `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*. For more information, see *About Dependency Injection*.

   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 Pre*xxx* customization method as an anonymous method in the AddMethod call because Siebel Open UI creates this anonymous method for every presentation model instance 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 Applets*.

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

```
returnStructure[ "CancelOperation" ] = 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*.
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.ExecuteMethod("SetActiveControl", null);
```

   This code sets the active control to null. For more information, see *Disabling Automatic Updates* and *SetActiveControl Method*.
6. Add the property that Siebel Open UI uses to store the set of records that are pending deletion:

```
this.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.Get("DeletionPendingSet");
var currentSelection = this.Get("GetSelection");
if(value === "Y"){
 delObj[currentSelection] = this.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 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 List Applets*.

   After examining these properties, assume you determine 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 determine that you can use the GetRecordSet property to get this full set of records.
8. Save the recyclebinpmodel.js file.

About Dependency Injection

Dependency injection is a software development 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 it 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* for dependency injection. For an example that uses dependency injection, see *Customizing the Physical Renderer to Refresh the Carousel*.

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*, *SetProperty Method*, and *SetActiveControl Method*.

ExecuteMethod calls a method that the presentation model references. 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 a presentation model references. For more information, see *About Dependency Injection* and *ExecuteMethod Method*.

Customizing the Presentation Model to Delete Records

This task is a step in *Process of Customizing the Presentation Model*.

The following figure 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.

```
RecycleBinPModel.prototype.Init = function(){
    SiebelAppFacade.RecycleBinPModel.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);
```

To customize the presentation model to delete records

1. In the recyclebinpmodel.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. For this example, 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* and *DeleteRecord Method*.

   This configuration is similar to the configuration you added in Step 1 in the topic *Creating the Presentation Model* 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.
3. Set the InDeletion property to true only if Siebel Open UI starts the deletion process:

   ```
   this.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.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.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.

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:

For more information, see *Disabling Automatic Updates* and *SetActiveControl Method*.

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

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*. In this example, this flow indicates that you must use the HandleRowSelect method. The presentation model that Siebel Open UI uses for list applets references HandleRowSelect, so you can configure Siebel Open UI to use ExecuteMethod to call it. For more information, see *HandleRowSelect Method*.

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 another method to make sure that it can call this other 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 that resides 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 the 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 *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*.

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", []);
```

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*.
    - `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.

12. Set the DeletionPendingSet property to zero:

```
this.SetProperty("DeletionPendingSet", []);
```

    This code sets the DeletionPendingSet property to zero after Siebel Open UI finishes running 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*.

14. Set the InDeletion flag to false:

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

    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 it exits the loop during a synchronous request, then it sends all DeleteRecord requests sequentially. In this situation, it 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 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 recyclebinpmodel.js file.

About Synchronous 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.

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 is 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.

Overriding Predefined Methods in Presentation Models

This task is a step in *Process of Customizing the Presentation Model*.

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 not find the field in the list of fields that it uses, and the client creates an error. To avoid this situation, in this topic you 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. For more information, see *GetFormattedFieldValue Method*.

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 *Flows That Create New Records in List Applets, Updating the User Interface*.
2. In the recyclebinpmodel.js file, configure Siebel Open UI to conditionally override and customize the method:

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

   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*.

3. Save the recyclebinpmodel.js file.

Customizing the Presentation Model to Handle Notifications

This task is a step in *Process of Customizing the Presentation Model*.

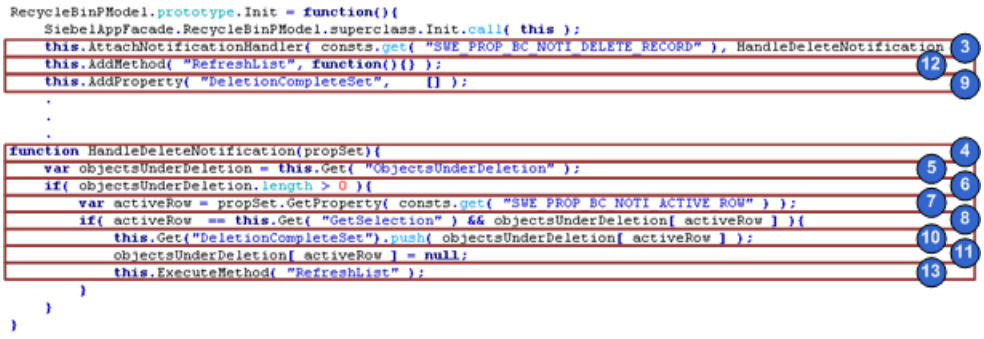
The Siebel Server sends a record deletion confirmation when it receives the InvokeMethod request for the DeleteRecord method. You can write a handler for the NotifyDeleteRecord notification to process this confirmation in the client. For more information, see *DeleteRecord Method*.

Siebel Open UI packages the 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 that resides on the server, then the following work occurs:

1. Siebel Open UI sends a NotifyDeleteRecord notification to the client.
2. The client sends a request to the server.
3. The server processes the request.
4. 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.
5. If the client 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* .

The following figure 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.



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* 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 that the presentation model references 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*.
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 Siebel Open UI has flagged for deletion:

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

   You configured this property in Step 10 in the topic *Customizing the Presentation Model to Delete Records* 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 then 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 that is 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*.

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 whether or not 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 Requests*.

- 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("DeletionCompleteSet", []);
```

10. Store the deleted record:

```
this.Get("DeletionCompleteSet").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 in Step 9.

11. Remove the record from the Deletion in Progress list:

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

12. Add the RefreshList method:

```
this.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*. For more information, see *How Siebel Open UI Uses Nondetailed Data to Indicate Modifications That Occur in Detailed Data*.

13. Run the RefreshList method:

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

14. Save the recyclebinpmodel.js file.

How Siebel Open UI Uses Nondetailed Data to Indicate Modifications

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 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.

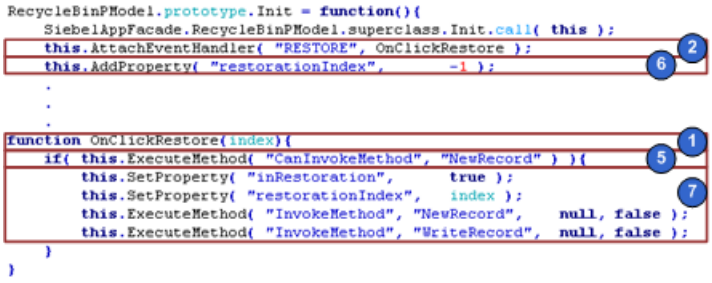
This example uses the RefreshList method as an indicator that Siebel Open UI modified something in the DeletionCompletedSet property. When you configure the physical renderer in *Customizing the Physical Renderer to Refresh the Carousel*, you configure Siebel Open UI to use the AttachPMBinding method to bind a physical renderer method to the RefreshList method. You also configure it 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*.

Attaching an Event Handler to a Presentation Model

This task is a step in *Process of Customizing the Presentation Model*.

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.

The following figure 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.



To attach an event handler to a presentation model

1. In the recyclebinpmodel.js file, add the method that handles the event:

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

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

   ```
   On
   ```

   Siebel Open UI calls this method when the user clicks the plus sign (+).
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 this 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* and *About Dependency Injection*.

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*. Note that Siebel Open UI uses the NewRecord method, and then uses 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*.

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 Form Applets*. 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*.

   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.ExecuteMethod("CanInvokeMethod", "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.AddProperty("restorationIndex", -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 then Siebel Open UI stores this value in the restorationIndex property of the presentation model.

7. Configure the OnClickRestore method:

   ```
   this.SetProperty("inRestoration", true);
   this.SetProperty("restorationIndex", index);
   this.ExecuteMethod("InvokeMethod", "NewRecord", null, false);
   ```

```
this.ExecuteMethod("InvokeMethod", "WriteRecord", null, false);
```

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 write this customization in the next topic in this example, *Customizing Methods in the Presentation Model to Store Field Values*.

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.

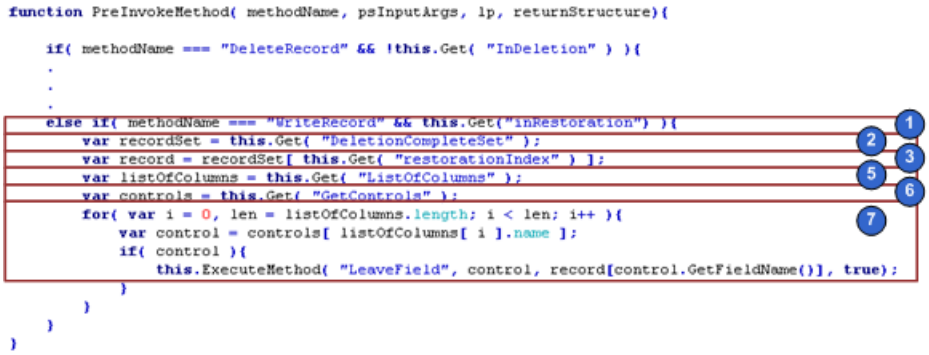
8. Save the recyclebinpmodel.js file.

Customizing Methods in the Presentation Model to Store Field Values

This task is a step in *Process of Customizing the Presentation Model*.

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.

The following figure 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.



To customize methods in the presentation model to store field values

1. In the recyclebinpmodel.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*.
2. Get the set of records for the recycle bin:

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

   In Step 10 in the topic *Customizing the Presentation Model to Handle Notifications*, 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 in the topic *Attaching an Event Handler to a Presentation Model*.
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*. Note that Siebel Open UI calls the LeaveField method as the last step in the flow. This 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 *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*.
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 that 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, it 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*.
- `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*.

8. Save the recyclebinpmodel.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*.

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* and *Customizing Events*.

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

1. In the recyclebinpmodel.js file, add the following code to the Init method:

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

   You use the Init method to send a WriteRecord call to the Siebel Server. For more information, see *WriteRecord Method* and *AttachPostProxyExecuteBinding Method*.
2. Add the following code anywhere in the recyclebinpmodel.js file:

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

```
}
```

where:

`PostWriteRecord` does the following work:

- Removes the record that Siebel Open UI restored in Step 7 in the topic *Customizing Methods in the Presentation Model to Store Field Values*. 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 to remove the record from the recycle bin in the client.
- Sets the inRestoration property of the presentation model to `false`. You set this property to `true` in Step 7 in the topic *Customizing Methods in the Presentation Model to Store Field Values* 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 recyclebinpmodel.js file.

Process of Customizing the Physical Renderer

This task is a step in *Roadmap for Customizing Siebel Open UI*.

To customize the physical renderer, do the following tasks:

1. *Setting Up the Physical Renderer*
2. *Customizing the Physical Renderer to Render the Carousel*
3. *Customizing the Physical Renderer to Bind Events*
4. *Customizing the Physical Renderer to Bind Data*
5. *Customizing the Physical Renderer to Refresh the Carousel*
6. *Modifying CSS Files to Support the Physical Renderer*

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*.

Setting Up the Physical Renderer

This task is a step in *Process of Customizing the Physical Renderer*.

The following figure illustrates the code that 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.



To set up the physical renderer

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

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\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*. For more information about the language_code, see *Languages That Siebel Open UI Supports*.
2. Use a JavaScript editor to open the recyclebinpmodel.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 sure 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*.
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("SiebelApp.Constants");
   ```
8. Create the class constructor:

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

9. Define the inheritance:

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

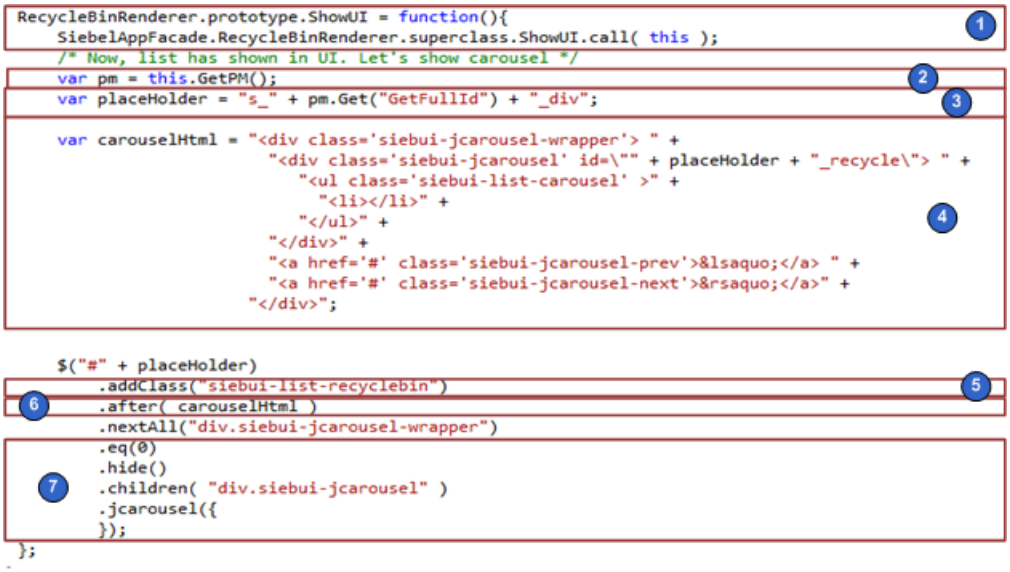
   For more information about inheritance, see *About Dependency Injection*.
10. Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Render the Carousel

This task is a step in *Process of Customizing the Physical Renderer*.

The ShowUI method of the JQGridRenderer physical renderer renders a list applet in the JQGrid control. This method places the third-party JCarousel control next to the grid. For more information, see *ShowUI Method*.

The following figure illustrates the code you use to customize the physical renderer to render a list applet. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



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*.
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 Document Object Model (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 is 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*.

4. Build the HTML for the third-party carousel plug-in:

```
var carouselHtml = "<div class='siebui-jcarousel-wrapper'> " +
 "<div class='siebui-jcarousel' id=\"" + placeHolder + "_recycle\"> " +
 "<ul class='siebui-list-carousel' >" +
 "<li></li>" +
 "</ul>" +
 "</div>" +
 "<a href='#' class='siebui-jcarousel-prev'>&lsaquo;</a> " +
 "<a href='#' class='siebui-jcarousel-next'>&rsaquo;</a>" +
 "</div>";
```

5. Add a CSS class:

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

6. Add the constructed HTML for the carousel after the carousel container:

```
.after(carouselHtml)
```

7. Modify the existing jcarousel div container, to make it a carousel:

   a. Locate the jcarousel div container in the first child of the parent container. The container will look similar to the following:

   ```
   .eq(0)
   .hide()
   .children( "div.siebui-jcarousel" )
   ```

   b. Make a carousel out of the jcarousel that you located in Step a:

   ```
   .jcarousel({
   });
   ```

8. Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Bind Events

This task is a step in *Process of Customizing the Physical Renderer*.

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 the HTML node that Siebel Open UI uses for the restore button.
- Styling changes that affect the appearance of the carousel based on user actions.

The following figure illustrates the code you use to customize the physical renderer to bind events to the carousel. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



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

- The carousel item, for every hover activity.
- The HTML node that Siebel Open UI uses for the Restore button.
- The Next and Previous icons in the carousel.

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*.
2. Identify the placeholder:

   ```
   var placeHolder = "s_" + this.GetPM().Get("GetFullId") + "_div";
   ```

3. Attach three event handlers for hover and click:

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

ShowRestoreButton is called when a user hovers on a carousel item, and HideRestoreButton is called when the hovering ends. If the user clicks the Add button, then AddFromRecycleBin is called.

4. Attach styling events to the Previous and Next buttons of the carousel:

```
$("#" + placeHolder + "_recycle")
 .parent()
 .find('.siebui-jcarousel-prev')
 .on('jcarouselcontrol:active', function () {
 $(this).removeClass('siebui-jcarousel-ctrl-inactive');
 })
 .on('jcarouselcontrol:inactive', function () {
 $(this).addClass('siebui-jcarousel-ctrl-inactive');
 })
 .jcarouselControl({
 target: '-=1'
 });

$("#" + placeHolder + "_recycle")
 .parent()
 .find('.siebui-jcarousel-next')
 .on('jcarouselcontrol:active', function () {
 $(this).removeClass('siebui-jcarousel-ctrl-inactive');
 })
 .on('jcarouselcontrol:inactive', function () {
 $(this).addClass('siebui-jcarousel-ctrl-inactive');
 })
 .jcarouselControl({
 target: '+=1'
 });
```

In this example, the first part of the code is finding the Previous button in the carousel container, and then attaching `jcarousel:active` and `jcarousel:inactive` events to it. When these events are triggered by the third-party plug-in, we call methods that set and unset styling classes on the buttons. Similarly, the styling classes are attached and removed for the Next button.

5. Define the handler methods:

   a. Use the following code to find the child for the add button and show it:

```
function ShowRestoreButton(evt) {
 if (evt && evt.currentTarget) {
 $(evt.currentTarget).children("a.siebui-citem-add").show();
 }

}
```

   b. Use the following code to find the child for the add button and hide it:

```
function HideRestoreButton(evt) {
 if (evt && evt.currentTarget) {
 $(evt.currentTarget).children("a.siebui-citem-add").hide();
 }
```

```
}
```

c. Use the following code to call the Restore method on the PM with the relevant index parameter

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

6. Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Bind Data

This task is a step in *Process of Customizing the Physical Renderer*.

The carousel in this example does not render data. Siebel Open UI only renders data in this example if it adds a record to 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*.
2. Save the recyclebinrenderer.js file.

Customizing the Physical Renderer to Refresh the Carousel

This task is a step in *Process of Customizing the Physical Renderer*.

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* and *AttachPMBinding Method*.

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.

The following figure illustrates the code you use to customize the physical renderer to refresh the carousel. 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" ];                                    6
}
SiebelJS.Extend( RecycleBinRenderer, SiebelAppFacade.JQGridRenderer );
RecycleBinRenderer.prototype.Init = function () {                                1
    SiebelAppFacade.RecycleBinRenderer.superclass.Init.call(this);
    this.AttachPMBinding( "RefreshList", RefreshCarousel );
};
.
.
.
function RefreshCarousel(){
    var pm          = this.GetPM(),
        recordSet   = pm.Get( "DeletionCompleteSet" ),                           2
        el          = $( "#s_" + pm.Get( "GetFullId" ) + "_div" + "_recycle" ),  3
        carousel    = el.data( 'jcarousel' ),                                    4
        count       = 0;
    carousel.reset();                                                            5
    for( var i = 0, len = recordSet.length; i < len; i++ ){
        if( recordSet[i] ){
            carousel
              .add( count,                                                       7
                    "<li>" + GetCurrentCarouselItems.call( this, recordSet[i],
                    this.listOfCols, i) + "</li>" );
            count++;
        }
    }
    carousel.size( count );                                                      8
    el.find( "a.siebui-citem-add").hide();                                       9
    el = carousel = null;                                                        10
}
```

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:

   ```
   this.AttachPMBinding("RefreshList", RefreshCarousel);
   ```

   In this example, you implemented the RefreshList method in the presentation model in Step 12 in the topic *Customizing the Presentation Model to Handle Notifications*. 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(),
    placeHolder = "s_" + pm.Get("GetFullId") + "_div",
    recordSet = pm.Get("DeletionCompleteSet"),
   ```

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

   ```
   el = $("#" + placeHolder + "_recycle"),
   ```

4. Construct the new mark-up:

```
for (var i = 0, len = recordSet.length; i < len; i++) {
 if (recordSet[i]) {
 markUp += "<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.

5. Determine the space that should be occupied by the grid, based on whether the carousel contains any records:

```
if (count > 0) {
 $("#" + placeHolder).addClass("siebui-span-md-10");
 el.parent().show().addClass("siebui-span-md-2");
}
else {
 $("#" + placeHolder).removeClass("siebui-span-md-10");
 el.parent().hide().removeClass("siebui-span-md-2");
}
```

   This step adds classes that decide the width of the original grid, effectively creating a fluid grid.

6. Add the newly constructed markup in Step 4, to the appropriate container:

```
el.children("ul.siebui-list-carousel").html(markUp);
```

7. Indicate to the plug-in that the content requires a reload:

```
el.jcarousel('reload');
```

8. Hide the restore button in the carousel:

```
el.find("a.siebui-citem-add").hide();
```

9. Remove the DOM references:

```
el = null;
```

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

10. Save the recyclebinrenderer.js file.

Modifying CSS Files to Support the Physical Renderer

This task is a step in *Process of Customizing the Physical Renderer*.

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

To modify CSS files to support the physical renderer

1. Open the CSS file, add the following code, and then save your changes:

```
.siebui-list-recyclebin {
 margin : 0px;
}
.siebui-jcarousel-wrapper {
 position: relative;
 height: 450px;
}
.siebui-jcarousel {
 position: relative;
 overflow: hidden;
 height: 100% !important;
 margin: 5px;
 width : 80%;
 border: 10px solid #fff;
 -webkit-border-radius: 5px;
 -moz-border-radius: 5px;
 border-radius: 5px;
 -webkit-box-shadow: 0 0 2px #999;
 -moz-box-shadow: 0 0 2px #999;
 box-shadow: 0 0 2px #999;
}
.siebui-jcarousel ul {
 width: 98%;
 position: relative;
 list-style: none;
 margin: 0;
 padding: 0;
}
.siebui-jcarousel ul li {
 margin-bottom : 5px;
}
.siebui-jcarousel-prev,
.siebui-jcarousel-next {
 transform: rotate(90deg);
 transform-origin: left top 0;
 float : left;
 position: absolute;
 width: 30px;
 height: 30px;
 text-align: center;
 background: #4E443C;
 color: #fff;
 text-decoration: none;
 text-shadow: 0 0 1px #000;
 font: 24px/27px Arial, sans-serif;
 -webkit-border-radius: 30px;
 -moz-border-radius: 30px;
 border-radius: 30px;
 -webkit-box-shadow: 0 0 2px #999;
 -moz-box-shadow: 0 0 2px #999;
 box-shadow: 0 0 2px #999;
}
 .siebui-jcarousel-prev {
 top : 0px;
 left : 45%;
}
 .siebui-jcarousel-next {
 top : 450px;
```

```
 left: 45%;
}
 .siebui-jcarousel-prev:hover span,
.siebui-jcarousel-next:hover span {
 display: block;
}
 .siebui-jcarousel-prev.inactive,
.siebui-jcarousel-next.inactive {
 opacity: .5;
 cursor: default;
}
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;
 -webkit-border-radius: 5px;
 -moz-border-radius: 5px;
 border-radius: 5px;
}
a.siebui-citem-add{
 display : block;
 top : 2px;
 right : 2px;
 float : right;
 width : 16px;
 height : 16px;
 background: url(../images/plus.png) no-repeat center center;
}
```

2. Add the CSS files that the third-party uses:

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

      `AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\3rdParty`

   b. Add the following subfolder hierarchy to the `3rdParty` folder:

      `\jcarousel\skins\tango\`

   c. Save the following files to the `tango` folder that you added in Step b:

      ```
      next-horizontal.png
      next-vertical.png
      prev-horizontal.png
      prev-vertical.png
      skin.css
      ```

      To get a copy of these files, see Article ID 1494998.1 on My Oracle Support. For more information about the CSS files and renderers that Siebel Open UI uses to render a list applet as a carousel, see *Customizing List Applets to Render as Carousels*.

3. Save the jquery.jcarousel.js file to the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\3rdParty
   ```

   Siebel Open UI uses this file to render a carousel. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

Process of Customizing the Plug-in Wrapper

This task is a step in *Roadmap for Customizing Siebel Open UI*.

To customize a plug-in wrapper, do the following tasks:

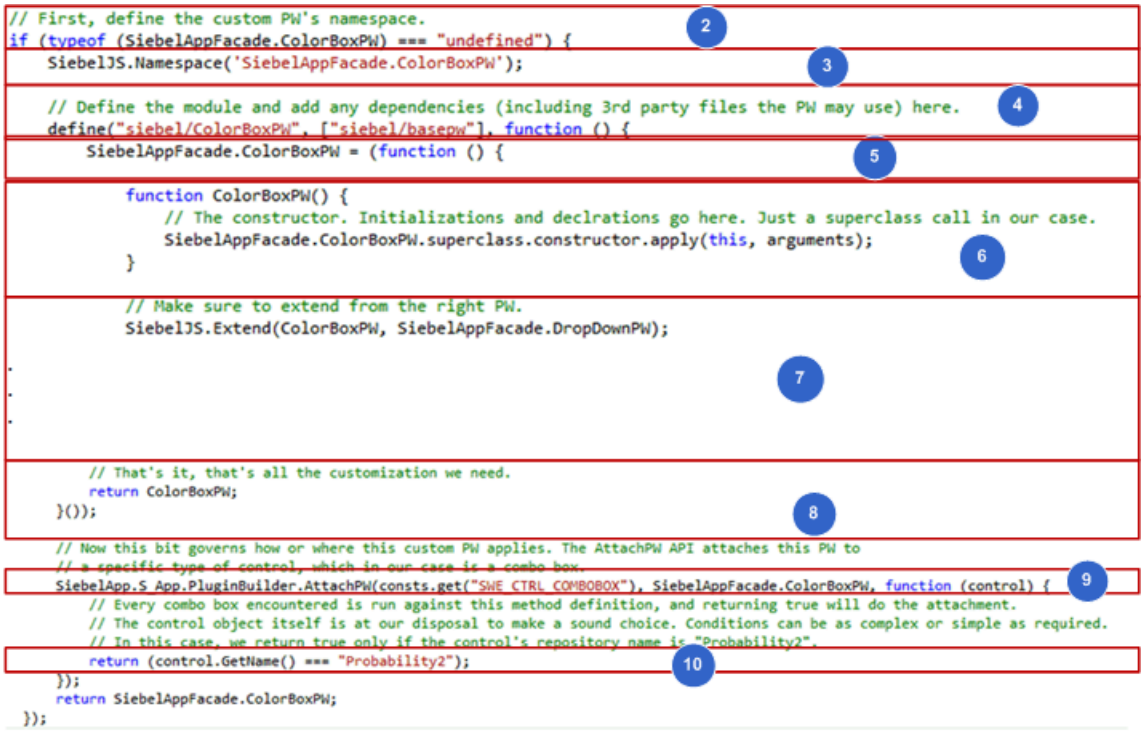
1. *Creating the Plug-in Wrapper*
2. *Customizing the Plug-in Wrapper to Display the Control Differently*
3. *Customizing the Plug-in Wrapper to Bind Custom Events to a Control*
4. *Customizing the Plug-in Wrapper to Define Custom Events*
5. *Customizing the Plug-in Wrapper to React to Value Changes of a Control*
6. *Attaching the Plug-in Wrapper to a Control Conditionally*

Creating the Plug-in Wrapper

This task is a step in *Process of Customizing the Plug-in Wrapper*.

The plug-in wrapper uses the Init method to configure the properties, methods, and bindings. For more information about these methods, see *Life Cycle of User Interface Elements*.

The following figure illustrates the code you use to create the plug-in wrapper. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



This topic describes how to modify code that resides in the ColorBoxPW.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 in this example. It also includes more comments that describe code functionality. To get a copy of this file, see Article1494998.1 on My Oracle Support.

For more information about the folders you can use to store your customizations, see *Organizing Files That You Customize*. For more information about the language_code, see *Languages That Siebel Open UI Supports*.

To create the plug-in wrapper

1. Create the plug-in wrapper file:
   a. Download a copy of the ColorBox.js file to the following folder:

      ```
      AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\custom
      ```

   b. Use a JavaScript editor to open the ColorBoxPW.js file that you downloaded in Step a.
2. Make sure the ColorBoxPW 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.ColorBoxPW) === "undefined"){
   ```

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

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

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

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

You must use the Define Method to ensure that 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*.

> **Note:** Any third-party files that the plug-in wrapper uses must be mentioned in the dependencies section of the define statement.

5. Define the class:

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

6. Define the class constructor:

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

7. Set up the injected dependency:

```
SiebelJS.Extend(ColorBoxPW, SiebelAppFacade.DropDownPW);
```

For more information about injected dependency, see *About Dependency Injection*.

8. Return the constructor:

```
return ColorBoxPW;
} ());
return SiebelAppFacade.ColorBoxPW;
});
```

9. Attach the plug-in wrapper:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_COMBOBOX"),
SiebelAppFacade.ColorBoxPW, function (control) {
```

10. Write the condition for which the plug-in wrapper should kick in:

```
return (control.GetName() === "Probability2");
```

11. Save the ColorBoxPW.js file.

Customizing the Plug-in Wrapper to Display the Control Differently

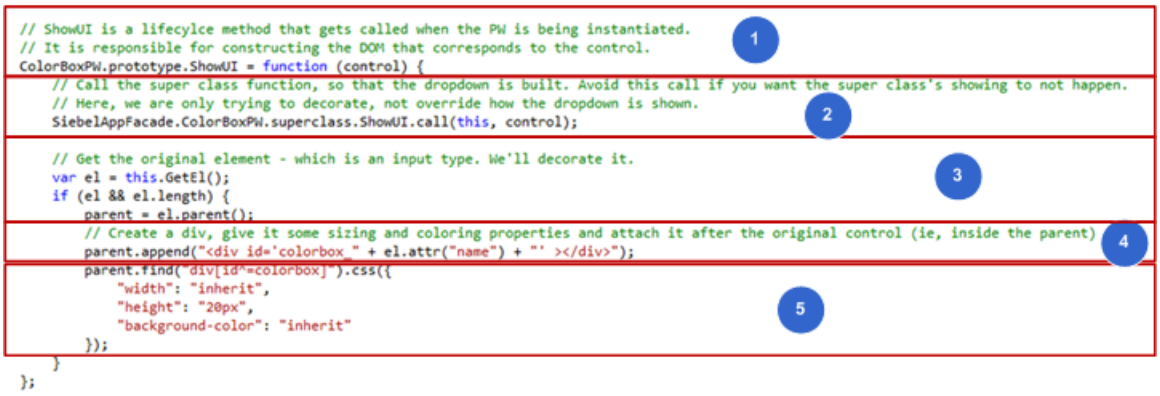
This task is a step in *Process of Customizing the Plug-in Wrapper*.

In this step, you customize the setup logic of the plug-in wrapper so that it adds a color-box to the control.

In this example, the ShowUI method is overridden to add a different element on to the DOM as a part of this control. The functionality of the control remains unaffected. Effectively, you will be decorating it with a new element.

This is an optional step: the base functionality of how a control looks and behaves can be completely changed based on your requirements. An out-of-the-box example of this type of modification is a flip switch that appears instead of a check box on touch devices in Siebel Open UI, which is accomplished using a plug-in wrapper.

The following figure illustrates the code you use to customize the ShowUI method of the plug-in wrapper. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



To customize the plug-in wrapper to display the control differently

1. In the colorboxpw.js file, introduce the ShowUI method that is a part of the life cycle of rendering a control.

   ```
   ColorBoxPW.prototype.ShowUI = function (control) {
   ```

2. Call the superclass method to get the dropdown to appear:

   ```
   SiebelAppFacade.ColorBoxPW.superclass.ShowUI.call(this, control);
   ```

   This will call the ShowUI method of the DropDownPW class, which is responsible for showing the drop down field in the Siebel Open UI client.

3. Get a reference to the existing element, and if it exists, get the parent element:

   ```
   var el = this.GetEl();
   if (el && el.length) {
   parent = el.parent();
   ```

   > **Note:** This step is required to position the new DOM element as a sibling to the current element.

   The GetEl() API framework method is a plug-in wrapper space that retrieves the jQuery element representing the control. parent() is a jQuery call which retrieves the parent node of the element in the DOM. For more information about the GetEl() API method, see *Architecture of Siebel Open UI*.

4. Add a new HTML div, which will serve as our color box:

   ```
   parent.append("<div id='colorbox_" + el.attr("name") + "' ></div>");
   ```

   You must specify a unique name for the element. In this example, `colorbox_` is added to the existing name of the original element. The `append()` and `attr()` specifications are both jQuery APIs. The former adds DOM elements at the end of a given element and the latter extracts the specified attribute.

5. Style the newly created div. This will serve as our colorbox:

   ```
   parent.find("div[id^=colorbox]").css({
   ```

```
"width": "inherit",
"height": "20px",
"background-color": "inherit"
});
```

The `css()` is a JQuery API that applies CSS styles to the given element. In this example, the colorbox gets the same width as the original dropdown and a height of 20 pixels. The original background color is inherited from the dropdown.

6. Save the ColorBoxPW.js file.

Customizing the Plug-in Wrapper to Bind Custom Events to a Control

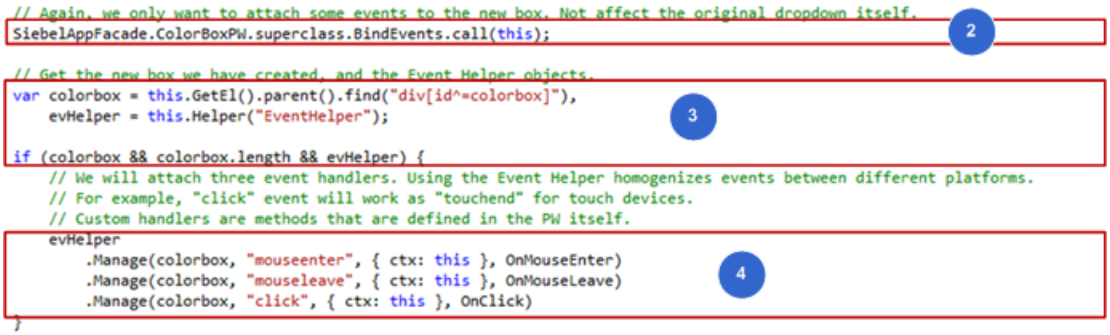
This task is a step in *Process of Customizing the Plug-in Wrapper*.

In this topic, you attach behavioral methods to the colorbox element that you created in *Creating the Plug-in Wrapper*.

In this example, the BindEvent method is overridden to attach custom handlers to a new element. The event handlers of the control remain unaffected, and the new element is decorated with some events.

This is an optional step: the base functionality of how a control looks and behaves can be completely changed based on your requirements. An out-of-the-box example of this type of modification is a flip switch that appears instead of a check box on touch devices in Siebel Open UI, which is accomplished using a plug-in wrapper.

The following figure illustrates the code you use to customize the BindEvents method of the plug-in wrapper. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



To customize the plug-in wrapper to bind custom events to a control

1. In the colorboxpw.js file, introduce the BindEvents method that is a part of the life cycle of rendering a control.

```
ColorBoxPW.prototype.BindEvents = function () {
```

2. Call the superclass method to attach the event handlers from the dropdown element:

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

This step calls the BindEvents of the DropDownPW class, which is responsible for attaching the events that the drop down field requires to operate correctly.

3. Get the element that was created and attached as a sibling to the actual dropdown element, and the Event Helper object:

```
var colorbox = this.GetEl().parent().find("div[id^=colorbox]"),
 evHelper = this.Helper("EventHelper");
 if (colorbox && colorbox.length && evHelper) {
```

   The Helper API is the framework method in the plug-in wrapper space that enables retrieving helper objects by name. For more information about the Helper API, see *Architecture of Siebel Open UI*.
4. Attach the required events to the new DOM element that was created. In this example, three handlers are attached to one element:

```
evHelper
 .Manage(colorbox, "mouseenter", { ctx: this }, OnMouseEnter)
 .Manage(colorbox, "mouseleave", { ctx: this }, OnMouseLeave)
 .Manage(colorbox, "click", { ctx: this }, OnClick)
```

   The Helper API is a method in the Event Helper object that takes the following four elements: the DOM element to which events should be attached, the event to be attached, the handler to be run, and other arguments. In this case, you are attaching one event for the each user hovering over the element, exiting the hover, and clicking on the element. For more information about the Helper API, see *Architecture of Siebel Open UI*.

Customizing the Plug-in Wrapper to Define Custom Events

This task is a step in *Process of Customizing the Plug-in Wrapper*.

In this topic, you define the behavioral methods that have been attached to the colorbox element that you created in you created in *Creating the Plug-in Wrapper*.

The following figure illustrates the code you use to define the handlers of the plug-in wrapper. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.

```
function OnMouseEnter() {
    // This is our handler for when the user hovers on the box. Let's inform them that it's clickable.
    $(this).append("<div id='info'>Click for Info...</div>");
}
function OnMouseLeave() {
    // This is our handler for when the user stops the hovering. Remove the information!
    $(this).find("#info").remove();
}
function OnClick() {
    // This is our handler for when the user takes the bait and clicks on the box.
    // We're going to construct a dialog that tells the user what the colors mean.

    // So first, we create a div with the correct html and attach it to the parent.
    var parent = $(this).parent(),
        html = "<div id='legend' title='Legend'>"
             + "<br><br>"
             + "<div style='width: 200px; height: 20px; background-color: rgb(255, 0, 0);'>  Do Not Pursue</div><br>"
             + "<div style='width: 200px; height: 20px; background-color: orange;'>  Pursue If Time Permits</div><br>"
             + "<div style='width: 200px; height: 20px; background-color: rgb(255, 255, 0);'>  Pursue</div><br>"
             + "<div style='width: 200px; height: 20px; background-color: rgb(0, 128, 0);'>  Pursue Aggressively</div><br>"
             + "</div>";
    parent.append(html);

    // Then we make the div into a jQuery-UI dialog; which puts the content into modal a popup.
    // More documentation about this api can be found in jQuery-UI docs.
    parent.find("#legend").dialog({
        resizeable: false,
        height: 275,
        width: 225,
        modal: true,
        buttons: {
            Cancel: function () {
                $(this).dialog("close");
            }
        }
    });
}
```

To define the event handlers for the plug-in wrapper

1. In the colorboxpw.js file, introduce the following private methods that will get called when the attached events occur on the element:

    a. The OnMouseEnter handler:

    ```
    function OnMouseEnter() {
     $(this).append("<div id='info'>Click for Info...</div>");
    }
    ```

    In this example, OnMouseEnter gets called when the `mouseenter` event occurs on the color box piece of the DOM. The context passed during the attachment of the events will be passed on to the handler method. Consequently, the `this` definition refers to the plug-in wrapper. In this example, a `div` is attached with an id of `info` that displays the following text: `Click for Info...`.

    b. The OnMouseLeave handler:

    ```
    function OnMouseLeave() {
     $(this).find("#info").remove();
    }
    ```

    This is the complementary method to the OnMouseEnter handler, and gets called when the `onmouseleave` event occurs on the color box DOM. This method removes the `div` that was previously added, consequently removing the display text.

    > **Note:** The two events will not run on touch devices, since they have no equitable actions.

2. Introduce the OnClick handler.

    Click is standardized by the event helper object to achieve uniformity across different devices. Consequently, it may be translated to different events based on the user's device. The click handler shows a popup that defines

the meaning of the different colors that the box can take on. In the first piece of the handler in this example, HTML built of a few styled divs and some corresponding text that forms the content of the information we are trying to show in the popup is constructed. The handler, and the content that is attached to the parent element are displayed here:

```
var parent = $(this).parent(),
html = "<div id='legend' title='Legend'>"
 + "<br><br>"
 + "<div style='width: 200px; height: 20px; background-color: rgb(255, 0,
0);'>  Do Not Pursue</div><br>"
 + "<div style='width: 200px; height: 20px; background-color:
orange;'>  Pursue If Time Permits</div><br>"
 + "<div style='width: 200px; height: 20px; background-color: rgb(255, 255,
0);'>  Pursue</div><br>"
 + "<div style='width: 200px; height: 20px; background-color: rgb(0, 128,
0);'>  Pursue Aggressively</div><br>"
 + "</div>";
parent.append(html);
```

3. Make the section into a popup.

   For this, use the jQuery-UI provided dialog() API. In this example, the element is located by id using find, and converted to a modal dialog box:

```
parent.find("#legend").dialog({
 resizeable: false,
 height: 275,
 width: 225,
 modal: true,
 buttons: {
 Cancel: function () {
 $(this).dialog("close");
 }
 }

});
```

   This sets properties for the popup and adds a cancel button that closes the popup.
4. Attach the required events to the new DOM element that we have created. Here we will attach three handlers on to this element.

```
evHelper
 .Manage(colorbox, "mouseenter", { ctx: this }, OnMouseEnter)
 .Manage(colorbox, "mouseleave", { ctx: this }, OnMouseLeave)
 .Manage(colorbox, "click", { ctx: this }, OnClick)
```

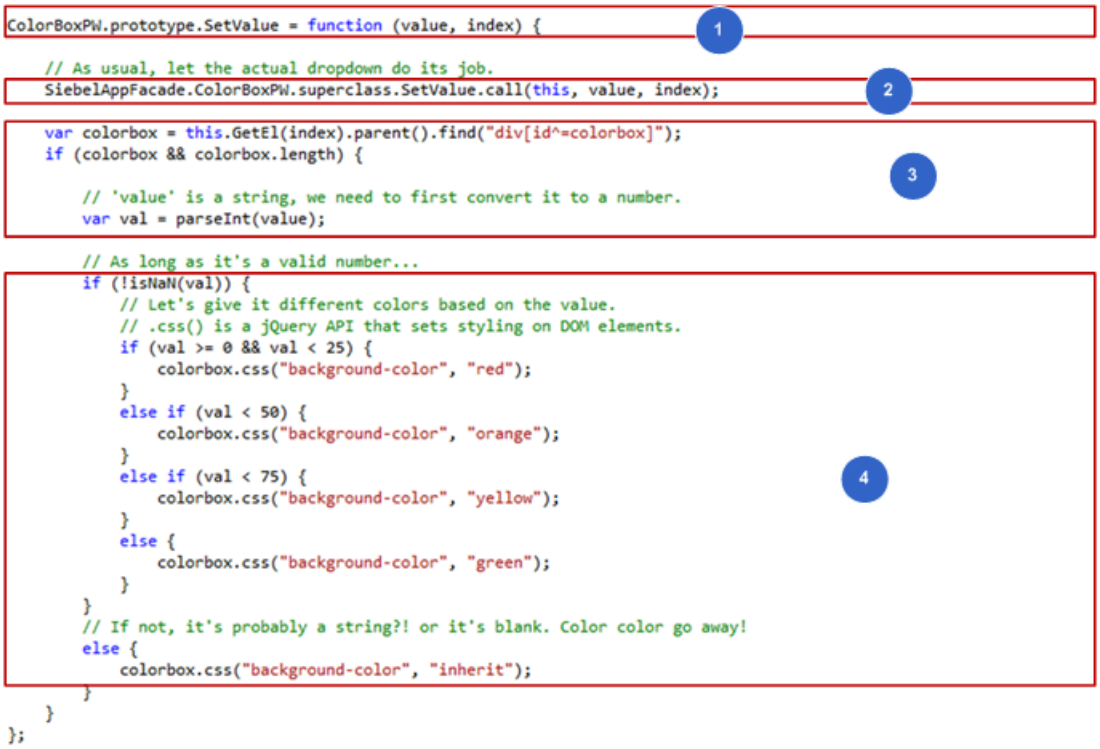
   The Helper API is a method in the Event Helper object that takes the DOM element in order to attach events. The attached event and the handler are deployed, along with other arguments.n this case, we are attaching one event each for the user hovering over the element, exiting the hover, and clicking on the element. For more information about the Helper API, see *Architecture of Siebel Open UI*.

Customizing the Plug-in Wrapper to React to Value Changes of a Control

This task is a step in *Process of Customizing the Plug-in Wrapper*.

In this topic, you define behavioral customizations when changes occur in a control value. These changes affect the appearance of the colorbox element that you created in *Creating the Plug-in Wrapper*.

The following figure illustrates the code you use to style the color box based on the value that is being set on a control. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



To define the value based modifications in the plug-in wrapper

1. In the colorboxpw.js file, introduce the SetValue method that is a part of the life cycle of a control's existence.

   ```
   ColorBoxPW.prototype.SetValue = function (value, index) {
   ```

   The SetValue API is called as part of a control life cycle when a value change occurs on the control, either directly by the user, or by the Siebel application. This call is responsible for the value change to appear in the DOM. In this example, SetValue is overridden in order to read into the value change that is happening on the control, and consequently makes modifications to the color box based on the value. For more information about the SetVAlue API, see *Architecture of Siebel Open UI*.
2. Call the superclass method to make sure that the dropdown receives the intended value:

   ```
   SiebelAppFacade.ColorBoxPW.superclass.SetValue.call(this);
   ```

   This will call the SetValue of the DropDownPW class, which is responsible for applying the correct value on to the dropdown field itself.
3. Get the new DOM element and the value that is being set:

   ```
   var colorbox = this.GetEl(index).parent().find("div[id^=colorbox]");
   if (colorbox && colorbox.length) {
   ```

```
var val = parseInt(value);
```

Because the value is in string form, and our future actions on this value involve treating it as a number, we need to convert it into a number form. The standard JavaScript method that is used for the purpose is parseInt.

4. Validate the value and specify values that modify the color box in different ways:

```
if (!isNaN(val)) {
 if (val >= 0 && val < 25) {
 colorbox.css("background-color", "red");
 }
 else if (val < 50) {
 colorbox.css("background-color", "orange");
 }
 else if (val < 75) {
 colorbox.css("background-color", "yellow");
 }
 else {
 colorbox.css("background-color", "green");
 }
}
else {
 colorbox.css("background-color", "inherit");
}
```

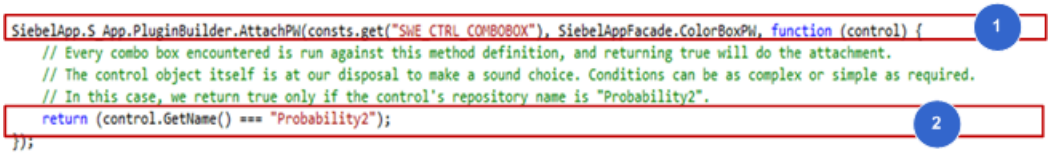
In this example, the value is verified to ensure that it is a number. If it is not, the background color is set to `inherit`, which sets the color to the same color as the dropdown element. This behavior would be applicable, for example, in cases where the user has entered a blank value, or inadvertently provided a string. If the value is a number, then use an if-else construct to define ranges and apply different colors on to the color box DOM element.

Attaching the Plug-in Wrapper to a Control Conditionally

This task is a step in *Process of Customizing the Plug-in Wrapper*.

This topic describes how to attach the plug-in wrapper you created in *Creating the Plug-in Wrapper* to a control.

The following figure illustrates the code you use to attach the plug-in wrapper to a control conditionally. Each number in this figure identifies the corresponding step number in the numbered task list that this book includes immediately after this figure.



To attach the plug-in wrapper to a control conditionally

1. In the colorboxpw.js file, introduce the AttachPW method from the PluginBuilder namespace that attaches the presently defined plug-in wrapper to a given type of control:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_COMBOBOX"),
```

```
SiebelAppFacade.ColorBoxPW, function (control) {
```

In this customization, the intention is to apply the plug-in wrapper to a dropdown type of control. To achieve this customization the SWE_CTRL_COMBOBOX is used for the dropdown type. All controls are customizable. With this customization, every dropdown encountered by the Siebel Open UI client will use this method.

2. Define the condition under which the attachment should occur, and to which specific instance of the control. The return value of the method used in Step 1 decides whether the plug-in wrapper attaches to a particular control. Returning true will mean a positive attachment.

```
return (control.GetName() === "Probability2");
```

Use the control object to create this condition. Since the intention is to attach the plug-in wrapper for all repository controls that have a name of `Probability2`, true will be returned when the name of the condition matches.

> **Note:** Plug-in wrappers are not restricted to any Presentation Model or Physical Renderer. Also, a customization defined on a plug-in wrapper will be applicable throughout the Siebel Open UI client, as long as the condition is satisfied. In this example, any control having a repository name of "Probability2" in any screen or view will be attached to this plug-in wrapper.

3. Define conditions for plug-in wrapper attachments. Conditions used can be as complex as necessary, based on the requirements. Use following examples as guidance for defining conditions:

```
Attach a plug-in wrapper to all TextArea fields in Opportunity List applet:
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_TEXTAREA"),
SiebelAppFacade.CustomPW, function (control) {
 return (control.GetAppplet().GetName() === "Opportunity List Applet");
});
```

   a. Attach a plug-in wrapper to all Date Fields in Contact Form applet and Account Form Applet:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_DATE_PICK"),
SiebelAppFacade.CustomPW, function (control) {
 var appletName = control.GetAppplet().GetName();
 return (appletName === "Contact Form Applet" || appletName === "Account Form
Applet");
});
```

   b. Attach a plug-in wrapper to a specific Text Box in a specific applet only:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_TEXT"),
SiebelAppFacade.CustomPW, function (control) {
 var appletName = control.GetAppplet().GetName();
 return (appletName === "Contact Form Applet" && control.GetName() === "Last
Name");
});
```

   c. Attach a plug-in wrapper to all Dropdowns in a particular application:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_COMBOBOX"),
SiebelAppFacade.CustomPW, function (control) {
 return (SiebelApp.S_App.GetName() === "Siebel EPharma")
});
```

   d. Attach a plug-in wrapper to all check boxes in a view when they are accessed on touch devices:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_CHECKBOX"),
SiebelAppFacade.CustomPW, function (control) {
 return (SiebelAppFacade.DecisionManager.IsTouch() &&
```

```
control.GetApplet().GetView().GetName === "Opportunity Detail View")
});
```

Configuring the Manifest for the Recycle Bin Example

This task is a step in *Roadmap for Customizing Siebel Open UI*.

This topic describes how to configure the manifest for the recycle bin example. For more information, see *Configuring Manifests*.

To configure the manifest for the recycle bin example

1. Make sure your presentation model and physical renderer use the define method.

   You do this in Step 4 in the topic *Creating the Presentation Model* for the presentation model and in Step 5 in the topic *Setting Up the Physical Renderer* for the physical renderer.
2. Log in to a Siebel client with administrative privileges.
3. Navigate to the Administration - Application screen, and then the Manifest Files view.
4. In the Files list, add the following files:

   ```
   siebel/custom/recyclebinrenderer.js
   siebel/custom/recyclebinpmodel.js
   siebel/custom/carouselrenderer.js
   3rdParty/jcarousel/skins/tango/skin.css
   files/theme-aurora.css
   ```

   The file that resides in the files folder is the predefined file that you use in this example.
5. Administer the manifest for the physical renderer:
   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	SIS Account List Applet

   c. 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

Field	Value
Level	1

d. In the Files list, add the following files:

```
siebel/custom/recyclebinrenderer.js
```

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

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

f. In the Object Expression list, add a record with no value in the Expression field.
g. In the Files list, add the following file:

```
siebel/custom/recyclebinpmodel.js
```

Configuring the Manifest for the Color Box Example

This task is a step in *Roadmap for Customizing Siebel Open UI*.

In this topic, you will configure the manifest for the color box plug-in wrapper example. For more information, see *Configuring Manifests*.

To configure the manifest for the color box example

1. Verify that your plug-in wrapper uses the define method.
2. Log in to the Siebel Open UI client with administrative privileges.
3. Navigate to the Administration - Application screen, and then the Manifest Files view.
4. In the Files list, add the following file:

```
siebel/custom/colorboxpw.js
```

5. Modify the manifest for the physical renderer:
   a. Navigate to the Administration - Application screen, and then the Manifest Administration view.
   b. In the UI Objects list, add a new record with the following values:

Field	Value
Type	Application
Usage Type	Common
Name	PLATFORM INDEPENDENT

c. 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

d. In the Files list, add the file that you created in Step 4.

```
siebel/custom/colorboxpw.js
```

Testing Your Modifications

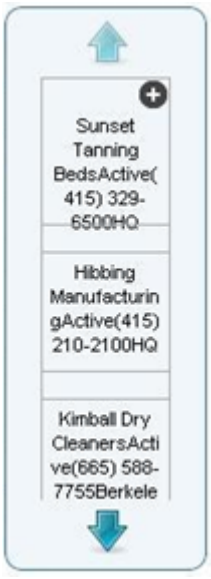
This task is a step in *Roadmap for Customizing Siebel Open UI*.

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.
3. Siebel Open UI deletes the records and adds them to the carousel recycle bin.

4. To restore a record, click the plus (+) icon in the carousel recycle bin (as shown in the following image).



5. Verify that Siebel Open UI recreates the record on the Siebel Server and adds it back to the Account list.
6. Navigate to the Opportunities screen, then to the Opportunities List view
7. Verify that the Probability field in the Opportunity form applet displays the color box and exhibits the correct behavior based on changes to values and clicks.

5 Customizing Styles, Applets, Fields, and Controls

Customizing Styles, Applets, Fields, and Controls

This chapter describes how to customize styles, applets, fields, and controls. It includes the following topics:

- *Customizing Logos, Themes, Backgrounds, Tabs, Styles, and Fonts*
- *Customizing Applets*
- *Customizing Controls*

Customizing Logos, Themes, Backgrounds, Tabs, Styles, and Fonts

This topic describes how to customize the logo, theme, background image, tabs, styles, and fonts that Siebel Open UI displays in the client. It includes the following information:

- *Customizing the Logo*
- *Customizing Themes*
- *Customizing the Synergy Theme*
- *Customizing the Aurora Theme*
- *Customizing Browser Tab Labels*
- *Using Cascading Style Sheets to Modify the Position, Dimension, and Text Attributes of an Object*
- *Adding Fonts to Siebel Open UI*
- *Specifying Font Families*
- *Customizing the Redwood Theme*

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

Siebel Open UI defines the logo that it displays in the client in CSS files. It uses the following predefined code to display the logo in the Aurora theme for screen sizes larger than 1199 pixels:

```
#_sweclient #_sweappmenu .siebui-logo
 float: left;
 height: 40px !important;
 line-height: 40px;
 background-image: url("../images/ebus.gif");
 background-repeat: no-repeat;
 background-origin: content-box;
 background-position: 4px 12px;
```

```
 width: 106px;
 white-space: nowrap;
}
```

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 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, the file might be my-logo.jpg.
2. Copy the file you created in Step 1 to the following folders:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\images\custom
   ```
3. Use an editor to open your custom CSS file that resides in the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\files\custom
   ```

   For example, open the my-style.css file.
4. Add the following code:

   ```
   #_sweclient #_sweappmenu .siebui-logo {
    background-image: url('../../images/custom/my-logo.jpg')
   }
   ```
5. (Optional) Modify the logo attributes, as necessary:
   a. Use an editor to open your custom CSS file.

      For example, open my-style.css.
   b. Add your custom code.

      Siebel Open UI uses the following predefined code to specify the logo attributes:

      ```
      #_sweclient #_sweappmenu .siebui-logo {
       float: left;
       height: 40px !important;
       line-height: 40px;
       background-image: url("../images/ebus.gif");
       background-repeat: no-repeat;
       background-origin: content-box;
       background-position: 4px 12px;
       width: 106px;
       white-space: nowrap;
      }
      ```

      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:

      ```
      #_sweclient #_sweappmenu .siebui-logo {
       width: 25px;
       height: 25px;
      }
      ```

6. Configure the manifest. For more information about how to do this step, see *Configuring Manifests*:

   a. Log in to a Siebel client with administrative privileges.
   b. Navigate to the Administration - Application screen, and then the Manifest Files view.
   c. Add the file that you modified in Step 4.

      For this example, you add the following file:

      ```
      custom/my-style.css
      ```

   d. Navigate to the Manifest Expressions view.
   e. In the Expressions list, add the following expression.

Field	Value
Name	`GRAY_TAB`
Expression	`LookupName (OUI_THEME_SELECTION, Preference ("Behavior","DefaultTheme")) = "GRAY_TAB"` In this expression, LookupName is a method that converts the language-dependent name of the theme to the language-independent name of the theme. Siebel Open UI uses the language-independent name.

   f. Navigate to the Manifest Administration view.
   g. In the UI Objects list, specify the following object.

Field	Value
Type	Application
Usage Type	Theme
Name	PLATFORM DEPENDENT

   h. In the Object Expression list, add the following subexpression.

Field	Value
Group Name	Leave empty.
Expression	Desktop

Field	Value
Level	1
Operator	Leave empty.
Web Template Name	Leave empty.

i. In the Object Expression list, add the following subexpression.

Field	Value
Group Name	Leave empty.
Expression	Enter the value that you specified in Step e.
Level	2
Operator	Leave empty.
Web Template Name	Leave empty.

j. Use the Move Up, Move Down, Indent, and Outdent buttons to rearrange the subexpressions, as necessary.
k. In the Files list, click Add.In the Files dialog box, click Query.
l. In the Name field, enter the following path and file name:

```
custom/my-style.css
```

m. Click Go.

7. Log out of the client, log back in to the client, and then verify that Siebel Open UI replaces the Oracle logo with your custom logo.

Customizing Themes

This topic includes an example that customizes the theme that Siebel Open UI displays in the client. It describes how to add a custom theme named Mobile Theme Gold that Siebel Open UI displays on a tablet.

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.

To customize themes

1. Create a new style sheet named theme-gold.css. Save this new file in the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\files\custom
   ```

   You can use any CSS file that includes your custom theme. You can also specify multiple CSS files. For this example, use theme-gold.css.
2. 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	Gold
Language-Independent Code	GOLD_THEME The value that you specify must match the theme name that you define in the manifest. In this example, this name is GOLD_THEME.
Parent LIC	NAVIGATION_TAB NAVIGATION_TREE NAVIGATION_SIDE

   e. For the new theme to be displayed only for desktop, then under Object Expression add a new record with `Expression = Desktop` and `Level = 1`. For the new theme to be displayed for all platforms (desktop, mobile, and so on), then under Object Expression add a new record with `Expression = null` and `Level = 1`.

3. Configure the manifest. For more information about how to do this step, see *Adding Custom Manifest Expressions*:

   a. Log in to a Siebel client with administrative privileges.
   b. Navigate to the Administration - Application screen, and then the Manifest Files view.
   c. Add the file that you created in Step 1.

      For this example, add the following file:

      ```
      files/custom/theme-gold.css
      ```

   d. Navigate to the Manifest Expressions view.
   e. In the Expressions list, add the following expression.

Field	Value
Name	`GOLD_THEME`
Expression	`LookupName (OUI_THEME_SELECTION, Preference ("Behavior", "DefaultTheme")) = "GOLD_THEME"` In this expression, LookupName is a method that converts the language-dependent name of the theme to the language-independent name of the theme. Siebel Open UI uses the language-independent name.

   f. Navigate to the Manifest Administration view.
   g. In the UI Objects list, specify the following object.

Field	Value
Type	Application This example configures Siebel Open UI to display your custom theme for the entire Siebel application. To specify this theme for a single object, see *Customizing Themes for Other Objects*.
Usage Type	Theme
Name	PLATFORM DEPENDENT

   h. In the Object Expression list, add the following subexpression.

Field	Value
Group Name	Leave empty.

Field	Value
Expression	Gold Theme If you must add a theme to some other platform, such as a phone or desktop, then specify this other platform. For example, specify Phone instead of Tablet.
Level	1
Operator	Leave empty.
Web Template Name	Leave empty.

i. In the Files list, add the file that you created in Step 1.

   For this example, you add the following file:

   ```
   files/custom/theme-gold.css
   ```

   You can use the Sequence field to determine the sequence that Siebel Open UI uses when it downloads cascading style sheets.

4. Test your modifications:

   a. Login to the Siebel Open UI client.
   b. Click User Preferences, click Behavior, and then click Edit.
   c. Verify that the Theme field includes the Gold value.
   d. Click Gold, and then click Save.
   e. Log out of the Siebel Open UI client, and then log back in.
   f. Verify that the Siebel Open UI client displays Gold theme.

Customizing Themes for Other Objects

This topic describes how to customize themes other objects and portlet applications.

To customize themes for other objects and portlet applications

- Do Step 1 through Step 4, except for Step 3, Step e, in the topic *Customizing Themes* and specify the object type and name of the object where Siebel Open UI must apply the style.

  For example, to apply the style only for an applet, set the Type to Applet, and the Name to the applet name, such as Contact List Applet.

  To specify the theme for another application, use the following expression:

  ```
  GetProfileAttr("PortletId") = "PtId"
  ```

  In this expression, *PtId* is the PtId argument of the URL to a Siebel portlet.

  For example:

  ```
  GetProfileAttr("PortletId") = "CRMOPTY1"
  ```

  For more information about PtId, see *Configuring Siebel Open UI to Consume Siebel Portlets*.

Customizing the Synergy Theme

This topic describes elements of the Synergy theme that can be customized. For more information about the Synergy theme, see *Siebel Fundamentals Guide* .

The Synergy theme is designed for Available Tab navigation only and is not suitable for Side Menu or Tree navigation.

Adding Landing Pages

By default, the landing page is enabled for all desktop applications. Follow the instructions in this topic to add a Synergy theme landing page for mobile applications.

To add landing page for mobile applications

1. Navigate to the Administration - Runtime Events screen, and then the Action Sets view.
2. Create an action set with a Name of your choosing and default field values.
3. Add a new action to the action set you created in Step 2 with the following defined values:

Field	Value
Name	*Action Name* For example, Landing Page.
Sequence	1
Profile Attribute	Is Landing Page Enabled

Field	Value
Value	TRUE

4. Navigate to the Administration - Runtime Events screen, and then the Action Sets view.
5. Create a run-time event with the following values:

Field	Value
Object Type	Application
Object Name	*Name of application* For example, Siebel Universal Agent.
Event	Login
Action Set Name	Select the action set that was created in Step 2.
Sequence	*Sequence number*

Removing Applets from Landing Pages

Follow the instructions in this topic to configure Synergy theme landing page.

To configure content on landing pages

1. Navigate to the Administration - Personalization screen, and then the Applets view.
2. Query for the applet that you want to remove from the landing page.
3. Add the following expression:

```
GetProfileAttr("Is SUI_THEME Landing View") = 'FALSE'
```

> **Note:** If there are any existing expressions, use the AND operator.

Removing Landing Pages

By default, in Siebel Open UI, desktop applications are configured to have landing pages, but Mobile applications do not have default landing pages. Follow the instructions in this topic to skip the display of landing page entirely.

To remove landing pages

1. Navigate to the Administration - Runtime Events screen, and then the Action Sets view.
2. Create an action set with a Name of your choosing and default field values.
3. Add a new action to the action set you created in Step 2 with the following defined values:

Field	Value
Name	*Action Name* For example, Remove Landing Page.
Sequence	1
Profile Attribute	Is Landing Page Enabled
Value	FALSE

4. Navigate to the Administration - Runtime Events screen, and then the Action Sets view.
5. Create a run-time event with the following values:

Field	Value
Object Type	Application
Object Name	*Name of application* For example, Siebel Universal Agent.
Event	Login
Action Set Name	Select the action set that was created in Step 2.
Sequence	*Sequence number*

6. Navigate to the Administration - Personalization screen, and then the Applets view.
7. Query for the Conditional Expression using containing the following string:

```
"*Is SUI_THEME Landing View*"
```

8. Select a record from the results of the search performed in Step 7, then remove the following expression from the Conditional Expression string:

```
GetProfileAttr('Is SUI_THEME Landing View') = 'FALSE'
```

9. Repeat Step 8 for every record returned in the search performed in Step 7.
10. Select Reload Personalization Rules from the applet menu.

Customizing the Aurora Theme

This topic describes how to customize the Aurora theme. For more information about the Aurora theme, see *Siebel Fundamentals Guide* .

To customize the Aurora theme

1. Create new CSS rules in mycustom.css and place the file in:

   ```
   AI_Install_Dir\applicationcontainer_external\siebelwebroot\files\custom
   ```

2. Navigate to the Sitemap and query for manifest and then select Manifest Files.
3. From the drop-down menu choose Name, then enter files in the text field and run the query.
4. Click the plus (+) icon in the menu bar.
5. In the name field, enter the path to the new CSS file, for example:

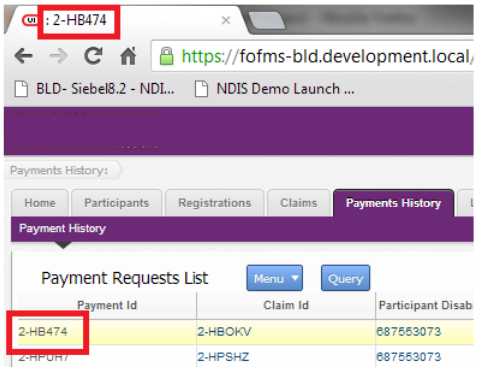
   ```
   files\custom\<mycustom.css>
   ```

6. Navigate to Manifest Administration and query for the Usage Type of Theme.
7. Click the plus (+) icon in the menu bar.
8. Create a new Platform Dependent record and click the plus (+) icon in Object Expressions.
9. Create an Aurora theme expression by clicking the plus icon (+) and entering Aurora in the Expression field.
10. Click the MVG icon.
11. In the Expressions pop-up window, click OK.
12. Set the new object expression's level to 1.
13. Click the plus (+) icon in the Files menu bar.
14. In the Files pop-up window, click the Search icon.
15. Enter files in the search field and click the Execute icon.
16. Select the check box for your new CSS file and click OK.
17. Navigate to Tools, User Preferences, and then Behavior.
18. From the Theme drop-down menu, choose Aurora.
19. Close the application and then restart it.

    The changes you made in the new CSS file are now active.

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 as shown in the following image:



If the label is not set for the view, the Id of the selected record is displayed by default. The tab name for the browser is set with the view Title or Title String Override, if it is defined within Siebel Tools. The same view Title is also looked up when the following script is called:

```
SiebelApp.S_App.GetActiveView().GetTitle().
```

Using Cascading Style Sheets to Modify the 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 the position, dimension, and text attributes of an object

1. Add these CSS rules to the end of your custom style sheet, my-style.css:

```
#_sweclient #_sweappmenu .PDQToolbarContainer {
 position: absolute;
 top: 40px;
 left: 610px;
 width: 140px;
}
#_sweclient #_sweappmenu .PDQToolbarContainer select {
 color: red;
 width: 140px;
}
```

2. Save the my-style.css file.
3. Verify that the Predefined Query drop-down list appear in the Help menu.

Adding Fonts to Siebel Open UI

This topic describes how to add custom fonts to Siebel Open UI. Although you can add custom fonts, it is recommended that your Siebel Open UI deployment use only Web-safe fonts because you might not be able to control font usage. For example, assume you deploy a custom font to all users in your company, and that you also add this font to Siebel Open UI. Assume that one of your Siebel Open UI users chooses this font in a text editor in Siebel Open UI, and then sends

this text in an email message to an external customer who has not installed this custom font on their computer. In this situation, your Siebel Open UI user can read the font but the external customer cannot read it.

Using Web-safe fonts helps to make sure that any browser or other client, such as a desktop computer or mobile device, can correctly render the text that your users provide, regardless of how each user configures font usage in their individual browsers or clients, or of the level of font customization that exists in your deployment environment. For more information about Web-safe fonts, see the topic that describes Web Safe Font Combinations at http://www.w3schools.com/cssref/css_websafe_fonts.asp.

To add fonts to Siebel Open UI

1. Create a JavaScript file that adds your custom font:

   a. Create a new JavaScript file named ckeditorfontadditions.js, and then save this file in the `custom` folder.

      For more information about this folder, see *Organizing Files That You Customize*.

   b. Add the following code to the file that you created in Step a. This code adds the fonts that Siebel Open UI displays in the Font picklists when the user edits text in the client:

```
if (typeof(SiebelAppFacade.CKEDITOREXTN) == "undefined") {
 Namespace('SiebelAppFacade.CKEDITOREXTN');
 (function() {
 SiebelApp.EventManager.addListner("postload", ckeditorextn, this);
 var updatedFont = "";
 function ckeditorextn() {
 try {
 if (CKEDITOR &&
 CKEDITOR.config.font_names !== updatedFont) {
 CKEDITOR.config.font_names = CKEDITOR.config.font_names +
 'font_families'
 updatedFont = CKEDITOR.config.font_names;
 }
 } catch (error) {
 // Nothing to do.
 }
 }
 }());
}
```

      where:

      - `CKEDITOR.config.font_names` is a predefined function that Siebel Open UI uses to store the list of fonts that it uses.
      - *font_families* specifies one or more font families that Siebel Open UI uses to render the font.
      - catch (error) catches any error that might occur when Siebel Open UI attempts to render the fonts that you specify. If an error occurs, then Siebel Open UI uses a predefined font to display the control.

      For this example, use the following code for *font_families*:

```
';Calibri/Calibri, Verdana, Geneva, sans-serif;'
```

      For more information about how to specify the font family, see *Specifying Font Families*.

2. Administer the manifest:

   For more information about how to do this step, see *Configuring Manifests*.

   a. Log in to the client as an administrator.
   b. Navigate to the Administration - Application screen, and then the Manifest Files view.
   c. In the Files list, add the file that you created in Step 2.

      You add the following record:

      ```
      siebel/custom/ckeditorfontadditions.js
      ```

   d. Navigate to the Administration - Application screen, and then the Manifest Administration view.
   e. In the UI Objects list, add a new record. Use values from the following table.

Type	Usage Type	Name
Application	Common	PLATFORM INDEPENDENT

   f. 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.

   g. In the Files list, add the file that you created in Step 2.

      You add the following record:

      ```
      siebel/custom/ckeditorfontadditions.js
      ```

   h. Refresh the manifest. Log out of the client, and then log back in to the client.

3. Verify that Siebel Open UI added your custom fonts:

   a. Navigate to the Administration Communications screen, and then the All Templates view.
   b. In the Compose Template section, in the Text window, click the Font drop-down, and then make sure the Font list displays the font that you specified in Step 1, Step b.

Specifying Font Families

You can use the following code to specify the font family:

```
function ckeditorextn() {
 try {
 if (CKEDITOR &&
 CKEDITOR.config.font_names !== updatedFont) {
 CKEDITOR.config.font_names = CKEDITOR.config.font_names +
 'font_families'
 updatedFont = CKEDITOR.config.font_names;
 }
 } catch (error) {
 // Nothing to do.
```

In this code sample, *font_families* specifies one or more font families that Siebel Open UI uses to render the font.

font_families can include one or more families. You must precede each font family with a semi-colon (;). For example:

```
;font_family_1;font_family_2;font_family_n
```

You must use the following format for each font family:

```
font_name/font_label,substitute_font_1,substitute_font_2, substitute_font_n,
generic_font_family
```

where:

- *font_name* specifies the name of the font, such as Calibri.
- *font_label* specifies the text label. It displays this label in the Font picklists in the client.
- *substitute_font_1* specifies the font if the font that *font_name* specifies does not exist in the client computer.
- *substitute_font_2* specifies the font if the font that *substitute_font_1* specifies does not exist in the client computer.
- *generic_font_family* specifies the font family if the font that *substitute_font_n* specifies does not exist in the client computer. Siebel Open UI chooses a font from this generic font family.

It is recommended that you specify a substitution font that resembles the font that it substitutes. For example, Calibri is a sans-serif, proportionally spaced font. If you specify Calibri as the *font_name*, then it is recommended that you specify a close approximation to Calibri for *substitute_font_1*, such as Verdana, which is also a sans-serif, proportionally spaced font. It is recommended that you use this same approach when you specify the remaining substitution fonts. For example, specify Geneva for *substitute_font_2*.

Consider the following example:

```
';Calibri/My Font, Verdana, Geneva, sans-serif;'
```

This code configures Siebel Open UI to do the following:

- Adds Calibri to the list of fonts that Siebel Open UI displays in Font picklists.
- Uses My Font as the label for the Calibri font that Siebel Open UI displays in Font picklists.
- If Calibri is not installed on the client computer, then Siebel Open UI uses the following sequence to determine the font that it displays:
  a. Uses Verdana for My Font.
  b. If Verdana is not installed on the client computer, then it uses Geneva for My Font.

c. If Geneva is not installed on the client computer, then it uses any sans-serif font that is installed on the client computer for My Font.

If you specify a font that includes a space character, then you must use double-quotes to enclose the entire font name. For example, you must use double quotes to enclose Times New Roman and Courier New:

```
';"Times New Roman"/My Font,Georgia,"Courier New",Serif;'
```

For more information about font families, see the topic that describes the CSS font family property at the W3 Schools website at http://www.w3schools.com/cssref/pr_font_font-family.asp.

Customizing the Redwood Theme

This topic describes elements of the Redwood theme that can be customized. For more information about the Redwood theme, see *Siebel Fundamentals Guide*.

Note: The Redwood theme is designed for Side Menu only and is not suitable for Tab or Tree navigation.

To customize the Redwood theme

1. Create new CSS rules in a custom css file, for example `mycustom.css` and place the file in: **SIEBEL_ROOT\applicationcontainer_external\siebwebroot\siebel\files\custom**
2. Navigate to the **Sitemap** and query for manifest and then select **Manifest Files**.
3. Click on the plus **(+)** icon in the menu bar.
4. In the name field, enter the path to the new CSS file, for example: **files\custom\<mycustom.css>**
5. Navigate to **Manifest Administration**.
6. Click the plus **(+)** icon in the menu bar.
7. Create a new Platform Dependent record and click the plus **(+)** icon in Object Expressions.
8. Enter **Redwood Theme** in the Expression field.
9. Set the new object expression level to 1.
10. Click the plus **(+)** icon in the Files menu bar.
11. In the Files pop-up window, click the **Search** icon.
12. Enter the custom CSS file which was created in Manifest Files view and click on the **Execute** icon.
13. Save the Record
14. Navigate to **Tools** > **User Preferences** > **Behavior**.
15. Choose **Redwood** from the **Theme** drop-down menu.
16. Close the application and then restart it.

    Note: The changes you made in the new CSS file are now active.

Customizing Applets

This topic describes how to customize applets. It includes the following information:

- *Displaying and Hiding Fields*

- *Allowing Users to Drag and Drop Data into List Applets*
- *Expanding and Collapsing Applets*
- *Customizing List Applets to Display a Box List*
- *Customizing List Applets to Render as Carousels*
- *Customizing List Applets to Render as Maps*
- *Customizing List Applets with Class Names*
- *Configuring List and Form Applets to display a color based on the value selected*
- *Disabling Oracle Maps*
- *Configuring the Focus in Siebel Applets*
- *Adding Static Drilldowns to Applets*
- *Allowing Users to Change the Applet Visualization*
- *Displaying Applets Differently According to the Applet Mode*
- *Adding Custom User Preferences to Applets*
- *Customizing Applets to Capture Signatures from Desktop Applications*
- *Customizing Applets to Capture Signatures for Siebel Mobile Applications*
- *Customizing Applets to Display Record Counts for Navigation Links*
- *Customizing Applets for Homepage Views in Redwood Theme*

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*.

This topic is similar to the *Displaying and Hiding Fields* topic, but with fewer details. It demonstrates how you can quickly modify a presentation model.

To customize the fields that are visible in an applet

1. Copy the JavaScript files:
   a. Download a copy of the partialrefreshpm.js file to the following folder:

      ```
      AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\custom
      ```

      For more information about this file, see *Text Copy of Code That Does a Partial Refresh for the Presentation Model*.
   b. Download a copy of the partialrefreshpr.js file to in the following folder:

      ```
      AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\custom
      ```

      For more information about this file, see *Text Copy of Code That Does a Partial Refresh for the Physical Renderer*.
2. Configure the manifest:
   a. Log in to a Siebel client with administrative privileges.
   For more information about the screens that you use in this step, see *Configuring Manifests*.

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

Field	Value
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.
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

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});
 this.AddMethod("FieldChange", OnFieldChange,{sequence : false, scope: this});
 };
 function SelectionChange(){
 var controls = this.Get("GetControls");
 var control = controls[ "JobTitle" ];
 var value = this.ExecuteMethod("GetFieldValue", control);
 this.SetProperty("ShowJobTitleRelatedField", (value ? true: false));
 }
 function OnFieldChange(control, value){
 if(control.GetName() === "JobTitle"){
 this.SetProperty("ShowJobTitleRelatedField", (value ? true: false));
 }
 }
 return PartialRefreshPM;
 }());
}
```

Text Copy of Code That Does a Partial Refresh for the Physical

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");
 //Module with its dependencies
 define("siebel/custom/partialrefreshpr", ["order!3rdParty/
jquery.signaturepad.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);
 // To act when FieldChange method is raised at PM level and execute our
custom code
 this.AttachPMBinding( "FieldChange", FieldChange );
 };

 function ModifyLayout(){
 var controls = this.GetPM().Get("GetControls");
 var control = controls[ "JobTitle" ];
 var value = this.GetPM().ExecuteMethod( "GetFieldValue", control );
 var canShow = ( value ? true : false);
 var WorkPhoneNum = controls[ "WorkPhoneNum" ];
 var FaxPhoneNum = controls[ "FaxPhoneNum" ];
 if(canShow){
 $( "#WorkPhoneNum_Label" ).parent().show(); // We need to take the parent
to get the whole div to hide
 $( "[name='" + WorkPhoneNum.GetInputName() + "']" ).parent().show();
 $( "#FaxPhoneNum_Label" ).parent().show();
 $( "[name='" + FaxPhoneNum.GetInputName() + "']" ).parent().show();
 }
 else{
 $( "#WorkPhoneNum_Label" ).parent().hide();
 $( "[name='" + WorkPhoneNum.GetInputName() + "']" ).parent().hide();
 $( "#FaxPhoneNum_Label" ).parent().hide();
 $( "[name='" + FaxPhoneNum.GetInputName() + "']" ).parent().hide();
 }
 }

 function FieldChange (control, value, index) {
 if( control.GetName() === "JobTitle" ){
 ModifyLayout.call(this);
 }
 }

 // We are overloading the standard PR ShowSelection to apply our customization
 // We ensure to first call the parent ShowSelection
 PartialRefreshPR.prototype.ShowSelection = function(index) {
 SiebelAppFacade.PartialRefreshPR.superclass.ShowSelection.call(this,index)
;
 ModifyLayout.call(this);
 };
 return PartialRefreshPR;
 } ());
 return "SiebelAppFacade.PartialRefreshPR";
 });
}
```

Allowing Users to Drag and Drop Data into List Applets

Note: Drag and drop functionality refers to the process of moving items from one place to another.

The example in this topic describes how to allow users to select and move data from a spreadsheet to the Contact List applet. You cannot use a calculated field value for the Client PM User Properties.

To allow users to select and move data into list applets

1. Modify the list 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. Create a spreadsheet:

   a. Open a spreadsheet application, such as Microsoft Excel.
   b. In the first row, add the following column headers for the columns that you must select and move: First Name, Last Name, Account, Mr/Mrs.

      - 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 after the column header row that you added in Step b. For example, add rows that include information about each contact as follows:

      - First Name: Antonia
      - Last Name: Pinas
      - Account: Partner PC Local
      - Mr/Ms: Ms

      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.

3. Identify the columns that you must select and move:

   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 an asterisk (red color) to indicate each required field. In the contact form, the Last Name and First Name fields are required.

4. Select and move the data:

   a. In the spreadsheet application, choose the cells that include the header and data information.
   b. Select and move the cells that you chose in Step a to the Contact List Applet in the Siebel application.

      Do the following to select and move 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 mouse button over the cursor.
      - Move 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.

Expanding and Collapsing Applets

This topic describes how to configure Siebel Open UI to display an applet as expanded or collapsed, by default.

To expand and collapse applets

1. Modify the applet:
    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.

       For example, query for SIS Account Entry Applet.
    d. In the Object Explorer, expand the Applet tree, and then click Applet User Property.
    e. In the Applet User Properties list, create two new applet user properties. Use values from the following table.

Name	Value
ClientPMUserProp	Default Applet Display Mode
Default Applet Display Mode	Use one of the following values: - **Expanded.** Siebel Open UI displays the applet in an expanded state, by default. - **Collapsed.** Siebel Open UI displays the applet in a collapsed state, by default.

    f. Compile your work.
2. Modify the Web template:
    a. Identify the Web template that you must modify, and then open it for editing.
    b. Add the following code:

```
<div od-type="form">
<div od-if="Web Engine State Properties, IsPrintOff">
 <div class="od-context-SelectStyle">
</div>
 <div class="siebui-collapsible-applet">
 <table datatable="0" summary="" width="100%" cellpadding="0"cellspacing="0"
border="0" align="center"
 <div class="siebui-collapsible-applet-header">
 <div od-include="CCTitle_Named"/>
 <div od-include="CCFormButtonsTop"/>
 <div od-type="error" type="Popup">
 <table datatable="0" summary="" class="od-context-Applet" width="100%"
cellpadding="0"
</div>
 <div class="siebui-collapsible-applet-content">
 <div od-type="form-applet-layout">
 </div>
```

```
<div>
<div>
<table>
<div>
```

where:

- `siebui-collapsible-applet` identifies the applet body.
- `siebui-collapsible-applet-header` identifies the section where Siebel Open UI adds the expand button or the collapse button.
- `siebui-collapsible-applet-content` identifies the section that Siebel Open UI expands or collapses.

3. Test your work:
   a. Log in to the client.
   b. Navigate to the applet that you modified in Step 1.
   c. Verify that Siebel Open UI displays the applet as expanded or collapsed according to the value that you set for the Default Applet Display Mode applet user property in Step 1.
   d. Verify that Siebel Open UI displays the expand and collapse button correctly.

      If Siebel Open UI expands the applet, then it must display the following collapse icon in the the applet:

      If Siebel Open UI collapses the applet, then it must display the following expand icon in the 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.

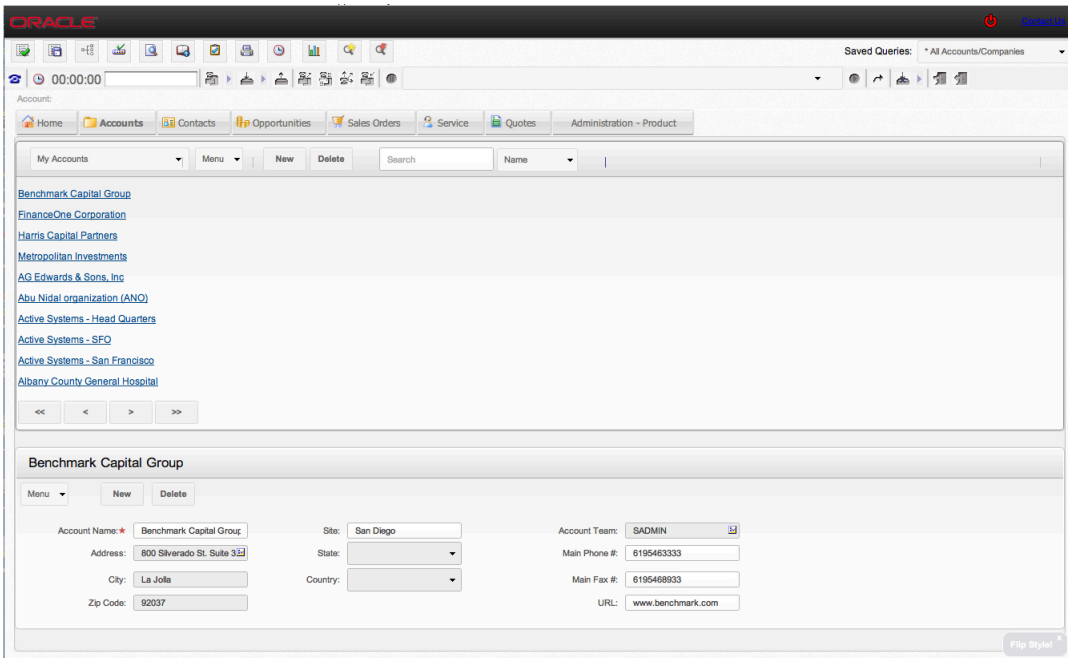
   For example, navigate to the Accounts screen, and then the Accounts list.

   Notice that Siebel Open UI displays the typical predefined list.
3. Open Windows Explorer, and then navigate to the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\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.



8. In Windows Explorer, navigate to the following folder:
   `AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\files\custom`
9. Use an editor to open the my-style.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.

Customizing List Applets to Render as Carousels

The example in this topic describes how to customize Siebel Open UI to render a list applet as a carousel in Siebel Call Center. To view different example carousel styles and to get the code for these styles, see the http://sorgalla.com/projects/jcarousel Web site.

To customize list applets to render as carousels

1. Add records in the client:
   a. Open the client, navigate to the Contacts screen, and then click the Contact List link.
   b. Add the following contact.

Field	Value
Last Name	Aamos
First Name	Ray

Field	Value

c. Click the link in the Last Name.
d. Click the Affiliations link.
e. In the Affiliations list, add four affiliations.
f. Make sure you choose a different value in the Account field for each record. Accept default values for all other fields.
g. Log out of the client.

2. Add the JavaScript files that Siebel Open UI uses to render the carousel:

a. Save the carouselrenderer.js file to the following folder:

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\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 for the business logic that it uses to display each user interface. The only difference is how Siebel Open UI renders each applet.

b. Save the jquery.jcarousel.js file to the following folder:

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\3rdParty\jcarousel
```

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. Add the CSS file that the third-party uses:

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

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\3rdParty
```

b. Add the following subfolder hierarchy to the `3rdParty` folder:

```
\jcarousel\skins\tango\
```

c. Save the skin.css file to the `tango` folder that you added in Step b:

To get a copy of this file, see Article ID 1494998.1 on My Oracle Support.

4. Add files to the manifest:

a. Log in to a Siebel client with administrative privileges.

For more information about the screens that you use in this step, see *Configuring Manifests*.

b. Navigate to the Administration - Application screen, and then the Manifest Files view.
c. In the Files list, add the following files. You must add a separate record for each file:

```
siebel/custom/carouselrenderer.js
3rdParty/jcarousel/skins/tango/skin.css
```

```
files/theme-aurora.css
```

Files that reside in the files folder are predefined files that you use in this example.

5. Administer the manifest for the applet:
   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	Pharma Professional Affiliation From List Applet

   c. 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

   d. In the Files list, add the following file:

```
siebel/custom/carouselrenderer.js
```

6. Administer the manifest for the Aurora theme:
   a. Navigate to the Manifest Expressions view.
   b. In the Expressions list, add the following expression.

Field	Value
Name	Aurora Theme
Expression	`LookupName (OUI_THEME_SELECTION, Preference ("Behavior", "DefaultTheme")) = "AURORA_THEME"`

Field	Value

c. Navigate to the Manifest Administration view.
d. In the UI Objects list, specify the following object.

Field	Value
Type	Application
Usage Type	Theme
Name	PLATFORM DEPENDENT

e. In the Object Expression list, add the following subexpression.

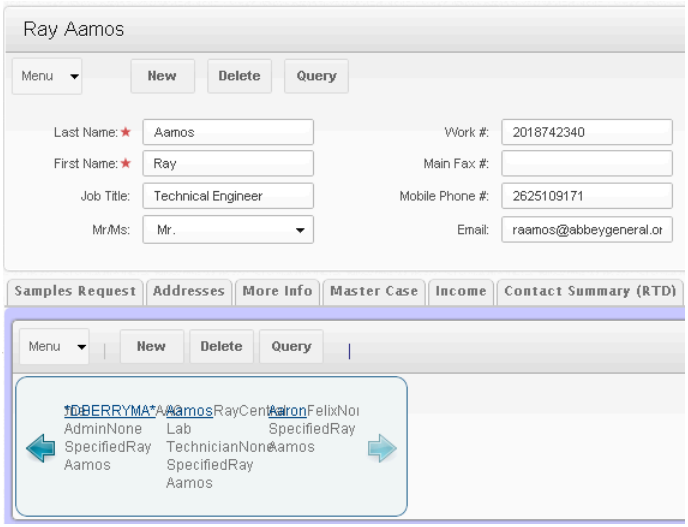
Field	Value
Group Name	Leave empty.
Expression	Aurora Theme
Level	1
Operator	Leave empty.
Web Template Name	Leave empty.

f. In the Files list, add the following files:

```
files/theme-aurora.css
3rdParty/jcarousel/skins/tango/skin.css
```

7. Test your modifications:

   a. Clear the browser cache.
   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 contains carousel data that runs together because no styling is defined for the carousel content. To fix this problem, continue to Step 8.



8. 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:

```
itemMarkup += "</span><br>";
```

   c. Modify the code you located in Step b to the following. You remove the break:

```
itemMarkup += "</span>";
```

   d. Use a JavaScript editor to open the skin.css file.
   e. Locate the following code:

```
.jcarousel-skin-tango .jcarousel-item {
 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:

```
.jcarousel-skin-tango .jcarousel-item {
 width: 318px;
 height: 75px;
}
```

   g. Locate the following code:

```
.jcarousel-skin-tango .jcarousel-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:

```
.jcarousel-skin-tango .jcarousel-item-horizontal {
 margin-left: 10;
 margin-right: 10px;
 color: black;
}
```

9. Test your modifications:
   a. Clear the browser cache.
   b. Refresh the view that you examined in Step 7.
   c. Make sure the styling no longer contains carousel data that overlaps, and that each record is displayed in its own block.



Customizing List Applets to Render as Maps

A list applet can be configured to display a map instead of a standard list of records. When the list applet is configured to display a map, the following features are available:

- **Markers.** A marker is displayed on the map at the location address for each record.
- **Contextual menu.** Clicking a marker reveals a contextual menu with the following options:
  - **View Details.** Clicking this option opens a pop-up dialog box with details about the record associated with the marker.
  - **Select.** Clicking this option zooms in on the map to the location address associated with the marker record.
- **Tooltip.** When you hover over a marker, a tooltip is revealed, showing the address associated with the record.
- **Map panning.**
- **Map zooming.**

The example in this topic describes how to customize Siebel Open UI to render a list applet as a map.

To customize list applets to render as maps

1. Add files to the manifest:
   a. Log in to a Siebel client with administrative privileges.

For more information about the screens that you use in this step, see *Configuring Manifests*.

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

c. In the Files list, add the following files. You must add a separate record for each file:

```
siebel/mappmodel.js
siebel/maprenderer.js
```

2. Administer the manifest for the applet:

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	Applet Name Where *Applet Name* is the name of the applet in which you want the map to appear.

c. 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

d. In the Files list, add the following file:

```
siebel/custom/maprenderer.js
```

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

Field	Value
Type	Applet
Usage Type	Presentation Mode
Name	Applet Name

Field	Value
	Where *Applet Name* is the name of the applet in which you want the map to appear.

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/mappmodel.js
```

3. Define the PM user properties:

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 modified in Step 2.
d. In the Object Explorer, expand the Applet tree, and then click Applet User Property.
e. In the Applet User Properties list, create four new applet user properties. Use values from the following table.

Name	Value
MapMarkerLocation	Business Component
MapMarkerTitle	Business Component
MapSelectedRowImage	The SVG image, as added in the CSS
MapUnSelectedRowImage	The SVG image, as added in the CSS

When specifying the Value field for the MapMarkerLocation and the MapMarkerTitle, the business component specified must meet at least one of the following conditions in order to properly display the markers on the map:

- It must be exposed in the list column of the list applet configured for the map.

- It must be exposed as an applet control or list column of a sibling applet to the map applet of the same business component.
- It must be set as a private field.

f. Compile your work.

4. Define the Web Template for the map:

a. Open Siebel Tools.

For more information about using Siebel Tools, see *Using Siebel Tools* .

b. In the Object Explorer, expand the Applet tree, and then click Applet Web Template.

c. 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 Map View to describe a map visualization.
Type	Edit List
Web Template	Choose a Web template that defines the desired visualization. For example, choose Applet Map.

d. Make sure Siebel Tools defines ODH for the Web template that you defined in Step c.

For example, make sure the Web Template Definition column in Siebel Tools includes ODH for Applet Map template. 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.

5. Test your modifications:

a. Clear the browser cache.

b. Refresh the list applet that you modified in Step 2.

c. Make sure that it renders a map.

Customizing List Applets with Class Names

You can use class names at the record level to customize list applets. Using the Applet PM user property you can define a class name based on a condition that is evaluated for each row in the list applet. Before you begin to customize a list applet:

- Configure a calculated field in a business component to use for applying the class name. Make sure the calculation produces an appropriate value that can be evaluated to produce the desired effect. For example, if you want to apply the class name when the calculated value is 1, 2, or 3, make sure that your calculation can handle all possible values it may encounter and produces the desired result.
- If the calculated field is not exposed in the UI, expose it as a business component private field.
- Configure the Applet PM User Property as follows (assuming the calculated field is named Record State):

- Name: Record State Field, Value: Record_State
- For each possible value for Record_State, create a name-value pair
  - Name: <State1>, Value <Class1>
  - Name: <Staten>, Value <Classn>
- Configure the Applet PM User Property to include all of the previously configured name-value pairs: Name: ClientPMUserProp, Value: Record State Field, State1, State2, ... Staten

For example, you can make a row in a list applet appear in a specific color if the revenue column has a value greater than $500. The following procedure explains how to implement this customization.

To customize a list applet to display a row in color based on the value of a specific column

1. Create a calculated field named "Record State" with the following calculated expression: [revenue > 500]
2. Expose the field as a Business Component Private field.
3. Configure the Presentation Model property as:
   a. Name: Record State field
   b. Value: Record State
4. For each expected value of the expression (in this example, true or false), add name-value pairs:
   a. Name: true
   b. Value: siebui-row-good-revenue
   c. Name: false
   d. Name: siebui-row-avg-revenue
5. Configure the properties to be exposed as Presentation Model properties:
   a. Name: ClientPMUserProp
   b. Value: Record State Field, true, false

   List applet rows having a revenue value greater than $500 have the siebui-row-good-revenue classname added. Rows that do not meet this criterion have the siebui-row-avg-revenue classname added. You can now add the a CSS definition to show the siebui-row-good-revenue rows in the spcified color if the revenue column has a value greater than $500.

Configuring List and Form Applets to display a color based on the value selected

Using the Applet Presentation Model (PM) user property you can configure a list applet column (s) or form applet control (s) to highlight a field with colors based on the value selected. You can add a CSS definition to show the column (s) or control (s) in the specified color.

For example, if the Customer Service Center wants to easily highlight customer ‘sentiment’ for Service Requests to assist Service Agents, then this is easily done. Where ‘sentiment’ is pre-defined - potentially via Oracle AI Services, then Developers can make a column in a list applet appear in a specific color (such as traffic lights) if the sentiment column has a value positive, negative, or neutral. This simple feature reduces development and configuration effort, whilst making the UX more intuitive for users.

To customize a List or Form Applets to display a color based on the value of a specific Calculated Field

1. Create a calculated field say<calc_field >, outcome of its value will work as a decision factor to display the background colour for the control.

   For example,

   ```
   IIf(LookupName("GENAI_SENTIMENT", [
   Sentiment])="POSITIVE","Positive",IIf(LookupName("GENAI_SENTIMENT",
   [Sentiment])="NEGATIVE","Negative",IIf(LookupName("GENAI_SENTIMENT",
   [Sentiment])="NEUTRAL","Neutral","")))
   ```

2. Expose the calculated field as a Business Component Private Field.

   > **Note:** If this BusComp User Prop already exists, add a comma, then your new field name, such as: <Existing Field 1>, <Existing Field 2>,Sentiment Indicator.

3. Set Force Active to TRUE.
4. Configure the properties to be exposed as Presentation Model properties:

   Name: ClientPMUserProp

   Value: Record State Field, Record State Control, Positive, Negative, Neutral

   (Positive, Negative, Neutral are the expected value of above calc field)
5. For each expected value of the expression (in this example, Positive, Negative or Neutral), add name-value pairs as applet user properties as mentioned in point b below.

   a. Name: Record State field

      Value: calc_field (Name should match as created in step 1)
   b. Below user properties Name i.e., point i. ii, iii or so on (should match exactly the outcome of calculated field created in step 1):

      i. Name: Positive

         Value: siebui-positive
      ii. Name: Negative

         Value: siebui-negative
      iii. Name: Neutral

         Value: siebui-neutral
   c. Name: Record State Control

      Value: Sentiment (comma separated BC field name. e.g., Sentiment, Priority, Status)

   Comma separated applet control (s) specified in the value of user property ‘Record State Control’, will have background colour based on the calculated field value. If calc field value = ‘Positive’, siebui-positive class name added. If calc field value = ‘Negative, siebui-negative class name added. If calc field value = ‘Neutral, siebui-neutral class name added.

Go to Site Map, Manifest Administration and associate the renderer to your applet for list applet.

Type	Usage Type	Name	File
Applet	Physical Renderer	<Applet Name>	Name `siebel/listcolumnattrpr.js`

Go to Site Map, Manifest Administration and associate the renderer to your applet for Form applet.

Type	Usage Type	Name	File
Applet	Physical Renderer	<Applet Name>	Name `siebel/formcontrolattrpr.js`

Customizing Form Applets with Icon Map Images

This topic describes how to configure icon map images to customize form applets.

To configure the icon map for other HTML Types (such as **Field**, **Text**, **Checkbox** and so on) you need to set the control as read-only in the Form applet. When you do this, in the user interface, irrespective of the HTML Type, the control is displayed with a similar functionality as **PlainText**.

Disabling Oracle Maps

Oracle Maps is configured as the default display for the Contact List Applet and the Contact Form Applet. You can disable the use of Oracle Maps by removing the configuration specified in *Customizing List Applets to Render as Maps*, or by using the following procedure.

To disable Oracle Maps

1. Run the Siebel application.
2. Navigate to Administration - Application, and then Manifest Application.
3. Query for the Contact List Applet in the Name field.
4. Inactivate all the fields having Group Name as Map in the "Object Expression" Applet.
5. Close the Siebel application.
6. In Siebel Tools, navigate to Applet Query for Contact Form Applet, Go to Control.
7. Deactivate the Show Route control.

Configuring the Focus in Siebel Applets

If you modify an applet, then you must make sure that your modification does not adversely affect how Siebel Open UI sets the focus in this applet. Siebel Open UI does the following work to set the focus in applets:

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 that describes 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 first to last (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 non-editable control that the list applet displays.
4. If Siebel Open UI does not find any non-editable 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.
- 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 non-editable 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 first-to-last 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 drill-down 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.

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.

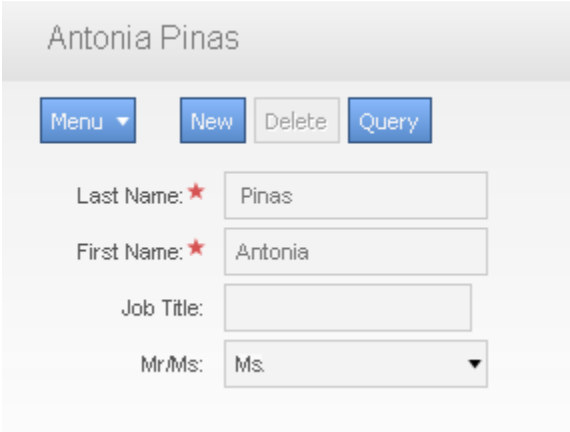
Property	Value
Field	Specify the same field that you specified in the Hyperlink Field property of the drill down object that you created in Step a.
HTML Type	Text

      For more information, see the topics about creating static drill down 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 drill down 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 (Pinas). If you complete 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.



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 = CCFMiscUtil_CreatePropSet();

inputPS.SetProperty("Key", "user_preference_name");
```

```
inputPS.SetProperty("user_preference_name", "user_preference_value");

pm.OnControlEvent(siebConsts.get("PHYEVENT_INVOKE_CONTROL"),
pm.Get(siebConsts.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. For more information, see *Adding Presentation Model Properties That Siebel Servers Send to Clients*.

2. Add the expand and collapse button:

a. Use a JavaScript editor to open the physical renderer that you edited in Step 1, Step a.

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 = CCFMiscUtil_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(siebConsts.get("PHYEVENT_INVOKE_CONTROL"),pm.Get(siebConsts.g
et("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-collapsible-applet-
content").show();
```

```
}
else
{
pm.SetProperty("Expand-Collapse", "Collapse");
//Write Code to collapse the applet
$("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapsible-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-collapsible-applet-
content").hide();
}
else
{
//Write Code to expand the applet
$("#s_" + pm.Get("GetFullId") + "_div").find(".siebui-collapsible-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-collapsible-
applet").append(expcolbtn);
```

For more information about how to edit HTML code for an applet, see *Customizing Logos, Themes, Backgrounds, Tabs, Styles, and Fonts*.

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 2, Step b.

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*
- *Configuring Siebel Open UI to Use Different Physical Renderers and Presentation Models According to the Applet Mode*

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 ODH:

- You use the predefined Applet List (Base/EditList) Web template that references the CCAppletList_B_EL Web template. This template uses an editable list layout.
- You add a new Edit Grid List Web template. This template 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:

      ```
      "Applet List (Base/EditList)"
      ```

   d. In the Object Explorer, select the Web Template, 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 CCAppletList_B_EL Web template 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: `EditGridList`

3. Modify the applet:

a. Do Step 1, 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 a 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*.
   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"

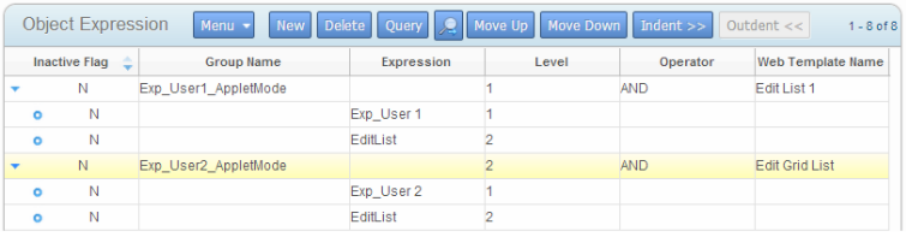
      For more information, see *GetProfileAttr Method*.
   d. Navigate to the Manifest Administration view.
   e. In the UI Objects list, specify the following applet.

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 configuration shown in the following table and image.

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
N	Exp_User1_AppletMode	<empty>	1	AND	Edit List 1
N	<empty>	Exp_User1	1	<empty>	<empty>
N	<empty>	EditList	2	<empty>	<empty>
N	Exp_User2_AppletMode	<empty>	2	AND	Edit Grid List
N	<empty>	Exp_User2	1	<empty>	<empty>

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
N	<empty>	EditList	2	<empty>	<empty>



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*.
   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:

```
siebel/custom/list_PM.js
```

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:

```
siebel/custom/list_PR.js
```

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:

```
siebel/custom/new_PM.js
```

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:

```
siebel/custom/new_PR.js
```

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.

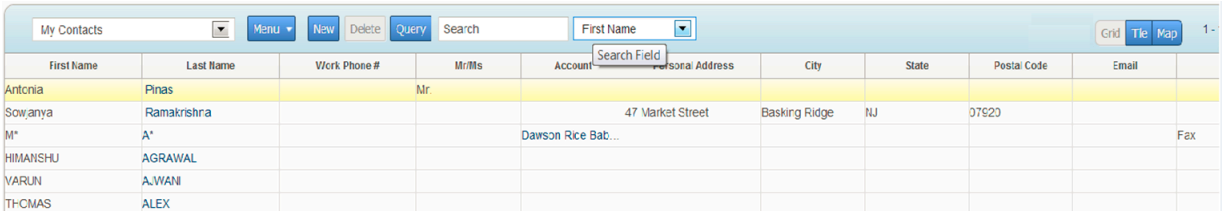
5. Test your modifications.

Allowing Users to Change the Applet Visualization

This topic describes how you can 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.

The following figure shows the Contacts List (that you modified in this topic) where the user can change the applet visualization. The user can click one of the applet visualization buttons (such as Tile, Grid, or Map) 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*.

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 Siebel Tools defines 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*.
   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: - 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 an ODH for the Web template that you defined in Step e.

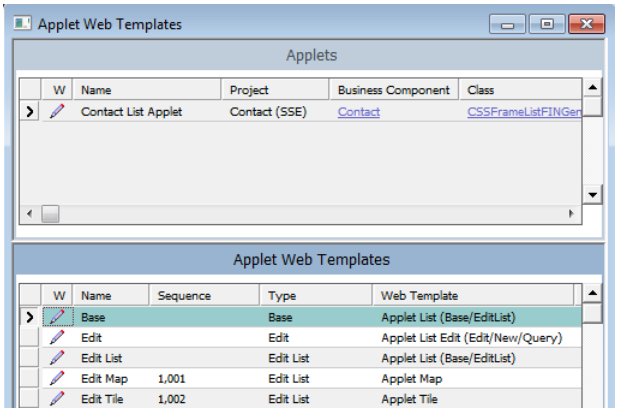
      For example, make sure the "Definition" column of Web Template in Siebel Tools includes ODH for the Applet Tile Web template. If your deployment requires a new Web template, then you must define it

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 configuration shown in the following image. As shown in this image, the Applet Web Templates associated with the Contact List Applet are:

- Base - Applet List (Base/EditList)
- Edit - Applet List Edit (Edit/New/Query)
- Edit List - Applet List (Base/EditList)
- Edit Map - Applet Map
- Edit Tile - Applet Tile



h. Compile your modifications.

2. Configure the manifest for the applet that you modified in Step 1:

a. Log in to a Siebel client with administrative privileges.

For more information about the screens that you use in this step, see *Configuring Manifests*.

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/Tilescrollcontainer.js

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

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

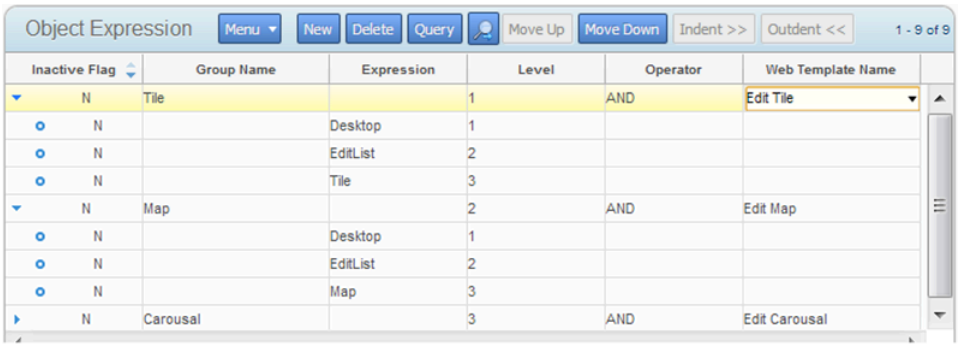
Field	Value
Type	Applet

Field	Value
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.

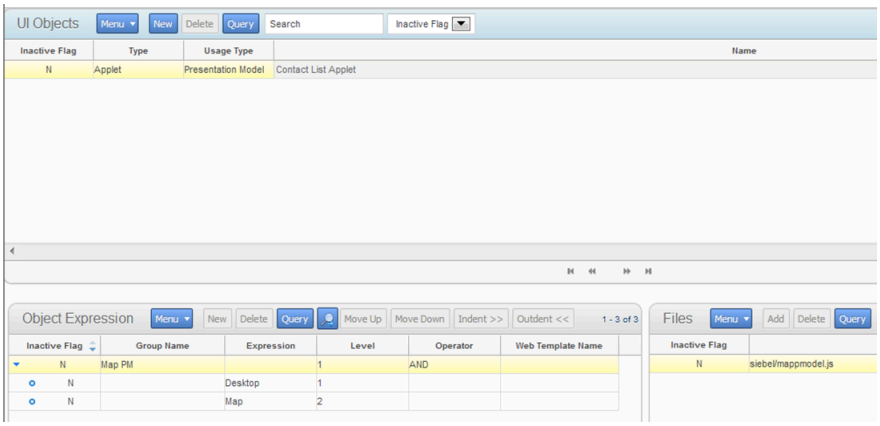
   Your completed work must resemble the following configuration. Use the Move Up, Move Down, Indent, and Outdent buttons to create the hierarchy. Note that you do not add files in the Files list for a Web

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*.



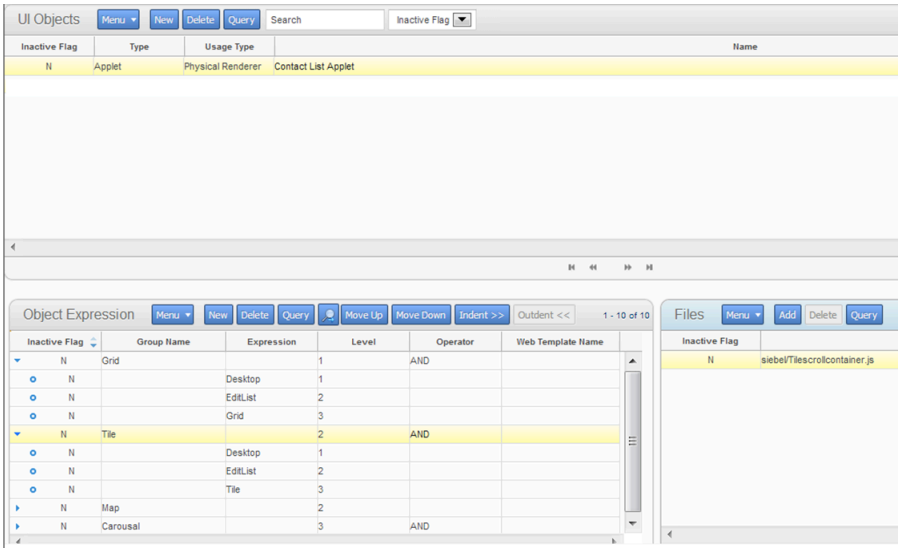
g. Configure the manifest for the presentation model for each applet visualization that you defined in Step 1.

   You add the UI object, object expressions, and files until the Manifest Administration screen resembles the following configuration.



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:



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 2, 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

Type	Display Value	Language-Independent Code

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 the first figure in this topic *Allowing Users to Change the Applet Visualization*.

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 the first figure in this topic *Allowing Users to Change the Applet Visualization* 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

The following table 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.

Visualization	Presentation Model	Physical Render	Web Template
Tile	Set Usage Type to Presentation Model. Set Name to List Applet Name.	Set Usage Type to Physical Renderer. Set Name to List Applet Name.	Set Usage Type to Web Template. Set the Name to Edit Tile.

Visualization	Presentation Model	Physical Render	Web Template
	Add the following to the Files list: `siebel/listpmodel.js`	Add the following to the Files list: `siebel/Tilescrollcontainer.js`	
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.

Auto Tile Visualization Feature

Tile Visualization mode is a way of viewing list applets within the Siebel application. With this feature, all qualifying list applets can be rendered as a tile automatically which can reduce the configuration effort. The tiles implementation (and the limitations of the tiles UI) remain the same and the individual usability of each applet when rendered as a tile has to be verified and must be addressed appropriately. Users do not need to configure the manifest and repository for each tile.

This topic describes the Auto Tile Visualization feature. It includes the following topics:

- *About Auto Tile Visualization of List Applets*
- *Criteria for Rendering Applets in Tile Mode*
- *Enabling Auto Tile Mode at the Application Level*
- *Enabling Auto Tile Mode at the List Applet level*
- *Predefined Manifest for Tiles Visualization*
- *Auto Tile Behavior*
- *Default Tile Content*
- *Customizing the Tile Content*
- *Customizing Tile Presentation for Qualifying List Applets*
- *Customizing Tile Presentation for a Specific List Applet*

- *Customizing the User Interface Layout of a List Applet*
- *Resolving Tile Presentation for a Specific List Applet*

About Auto Tile Visualization of List Applets

With the Auto Tile Visualization feature, the qualifying list applets can be rendered as a tile automatically. The tiles implementation (and the limitations of the tiles user interface) remains the same. Auto Tile Mode can be enabled by defining the application or the applet user property in Siebel Open UI.

- List applets must meet a specific set of criteria to have the tile mode enabled for them.
- Auto Tile can be enabled at the application level and can also be enabled or disabled for specific applets.
- There are no enhancements to the preexisting tiles usability.
- Auto Tile visual mode will render as similar to the preexisting tile.
- Individual usability of each applet when rendered as a tile has to be verified and needs to be taken care, if it doesn't meet the business requirements.
- The content, presentation and user interface layout of these tiles can be customized.

Criteria for Rendering Applets in Tile Mode

The qualifying list applets can be rendered in the Tile mode automatically. The list applets must conform to the following criteria to enable the Tile mode:

- Applet must be configured with `Type = 'Standard'`.
- Applet must be configured with `Applet Mode = Edit List` in the respective repository view.
- There must be at least one Applet Web Template with `Type = 'Edit List'`.
- Must not be configured as HIERARCHICAL list applet.
- Must not have the Applet Web Template which is configured against Tile/Map web template.

Note: For Siebel On Phone, all list applets are rendered in the Tile mode automatically.

Enabling Auto Tile Mode at the Application Level

The Auto Tile mode can be enabled at the application level for all qualifying list applets in the Siebel application. The example in this topic describes how to enable Auto Tile mode at the application level.

To enable Auto Tile mode at the application level

1. Open Siebel Tools.
   For more information, see *Using Siebel Tools* .
2. In the Object Explorer, click Application.
3. In the Applications list, query the Name property for the application that you must modify.
   For example, Siebel Universal Agent.
4. In the Object Explorer, expand the Application tree, and then click Application User Prop.
5. In the Application User Properties list, add the following application user properties.

Name	Value
EnableAutoTile	TRUE

Name	Value

6. Compile your modifications.

Enabling Auto Tile Mode at the List Applet level

The Auto Tile mode can be enabled at the List Applet level for all qualifying list applets in the Siebel application. The example in this topic describes how to enable Auto Tile mode at the list applet level (assuming that the application does not have the Auto Tile mode enabled) for a specific applet.

To enable Auto Tile mode at the list applet Level

1. Open Siebel Tools.

   For more information, see *Using Siebel Tools* .
2. In the Object Explorer, click Applet.
3. In the Applet list, query the Name property for the applet that you must modify.

   For example, SIS Account List Applet.
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 properties.

Name	Value
EnableAutoTile	TRUE

> **Note:** Set the value of the user property to False to disable the Auto Tile mode for a specific applet, assuming that the Auto Tile mode has been enabled at the application level.

6. Compile your modifications.

Predefined Manifest for Tiles Visualization

The Auto Tile feature needs manifest objects that cater to the Tile mode for the qualifying list applets. The required Manifest Objects are predefined, as shown in the following tables.

- The manifest object definition for UI Object (Applet Physical Renderer - DEFAULT LIST APPLET) is as follows:

Type	Usage Type	Name
Applet	Physical Renderer	DEFAULT LIST APPLET

The Object Expression for UI Object (Applet Physical Renderer - DEFAULT LIST APPLET) is as follows:

Expression	Level	File Name
Tile	1	`siebel/TileLayoutPR.js`

Expression	Level	File Name
Not Set	2	`siebel/jqgridrenderer.js`

- The manifest object definition for UI Object (Applet Web Template - DEFAULT LIST APPLET) is as follows:

Type	Usage Type	Name
Applet	Web Template	DEFAULT LIST APPLET

The Object Expression for UI Object (Applet Web Template - DEFAULT LIST APPLET) is as follows:

Expression	Level	Web Template Name
Tile	1	Tile

Auto Tile Behavior

The following table shows the behavior of Auto Tile at the application level or at the applet level, based on the application or applet user property values. It includes the Application User Property values, Applet User Property values, and a description of the relevant behavior.

Application User Property	Applet User Property	Behavior
`<not set>`	`<not set>`	No change from existing behavior.
`<not set>`	TRUE	Enables Tile mode only for the specified list applet.
`<not set>`	FALSE	No change from existing behavior.
TRUE	`<not set>`	Enables Tile mode for all the qualifying list applets in the application.
TRUE	TRUE	Enables Tile mode for all the qualifying list applets in the application.
TRUE	FALSE	Enables Tile mode for all the qualifying list applets except the specified list applets in the application.
FALSE	`<not set>`	No change from existing behavior.

Application User Property	Applet User Property	Behavior
FALSE	TRUE	Enables Tile mode only for the specified list applet (assuming it qualifies).
FALSE	FALSE	No change from existing behavior.

Default Tile Content

By default, the first nine list columns will be displayed in Tile mode.

- Applet Web Template Items under Applet Web Template (that is, with `Type = "Edit List"`) of qualified list applet, are sorted by their Item Identifier in ascending order.
- Based on this sort order, the first nine are chosen and the list columns which are mapped against these will be displayed in the Tile details.
- List columns are not displayed if they are not exposed or not selected on the list applet. In this case, fewer than nine columns would be displayed.

Customizing the Tile Content

Create the following applet user property to customize the details displayed when in tile mode.

Name	Value	Remarks
`ItemPrefForAutoTile`	<comma separated list of valid Item Identifiers> For example: `"501, 509, 503"`	None.
`ItemPrefForAutoTile:<AWT Type Mode>` For example, to configure the list of fields for Edit List Mode in Tile Visualization, use `ItemPrefForAutoTile:EditList`.	<comma separated list of valid Item Identifiers> For example: `"501, 509, 503"`	This change is applicable only for Siebel On Phone and view-based applets.

The list column restrictions include the following:

- Must be exposed and selected on the list applet. If none of the columns are mapped or selected against these, then the tile card will be empty.
- A maximum of nine web template items are used.
- Limited to the ones that are listed (so, fewer than nine means only that set).
- Any invalid (unmapped) identifiers are not displayed.

Customizing Tile Presentation for Qualifying List Applets

To customize the tile presentation for all qualifying list applets of an application, the following manifest objects must be changed to point to the custom JavaScript (JS) files.

The manifest object definition for UI Object (Applet Physical Renderer - DEFAULT LIST APPLET) is:

Type	Usage Type	Name
Applet	Physical Renderer	DEFAULT LIST APPLET

The Object Expression for UI Object (Applet Physical Renderer - DEFAULT LIST APPLET) changes to the following:

Expression	Level	File Name
Tile	1	`siebel/CustomTilePR.js`
Not Set	2	`siebel/jqgridrenderer.js`

Customizing Tile Presentation for a Specific List Applet

To customize the tile presentation for a specific list applet, the applet Manifest objects must be changed to point to the custom JS files. For example, the Opportunity List Applet is customized as shown here.

The manifest object definition for UI Object (Applet Physical Renderer - Opportunity List Applet) is:

Type	Usage Type	Name
Applet	Physical Renderer	Opportunity List Applet

The Object Expression for UI Object (Applet Physical Renderer - Opportunity List Applet) changes to the following:

Expression	Level	File Name
Tile	1	`siebel/CustomTilePR.js`
Not Set	2	`siebel/jqgridrenderer.js`

Customizing the User Interface Layout of a List Applet

The user interface layout for a list applet can be modified by making certain repository changes. The following table shows the repository changes to make in the Applet Web Template.

Field	Description
Name	Tile. The tile name must be Tile since it is used in the seeded manifest.
Type	Edit List
Sequence	This can be any other integer. For example: 100l.

Field	Description
Web Template	This can be any valid web template, where the layout definition supports tile. For example: Applet Tile.
Applet Web Template Items	Create the required applet Web template items, which must be displayed in the tile card under the newly created Tile applet Web template.

Resolving Tile Presentation for a Specific List Applet

The tile presentation may not be rendered correctly for certain list applets associated with the Custom Manifest Object definition. To resolve the tile presentation for a specific list applet, the applet Manifest objects must be changed to point to the custom JS files. For example, the Account Attachment Object Applet is customized as shown here.

The manifest object definition for UI Object (Applet Physical Renderer -Account Attachment Applet) is:

Type	Usage Type	Name
Applet	Physical Renderer	Account Attachment Applet

The Object Expression for UI Object (Applet Physical Renderer - Account Attachment Applet) is as follows:

Expression	Level	File Name
Not Set	1	`siebel/attachmentprenderer.js`

The default JS file must be added and the order level of the Account Attachment Applet must be modified.

The Object Expression for UI Object (Applet Physical Renderer - Account Attachment Applet) changes to the following:

Expression	Level	File Name
Tile	1	`siebel/TileLayoutPR.js`
Not Set	2	`siebel/attachmentprenderer.js`

Customizing Applets to Capture Signatures from Desktop Applications

A *signature capture* is an electronic capture of a user signature. This topic describes how to customize applets to capture signatures for calls in Siebel Open UI.

Note: This task uses Siebel Pharma as an example, but the procedure is similar when modifying a different application. For more information about migrating signatures from High Interactivity to Siebel Open UI, see the topic about configuring the digital migration service for signatures and the topic about rendering signatures in the user interface in *Siebel Life Sciences Guide* .

To customize applets to capture signatures for desktop applications

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 Call Signature Form Applet
      ```
   d. Right-click the applet you located in Step c, and then click Copy Record.
   e. Add an _PUI suffix to the name. For example:
      ```
      LS Pharma Call Signature Form Applet_PUI
      ```
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:

Name	Description
	- Set the Field property to Signature - Set the HTML Type property to InkData.
Disclaimer Text Signature Header Text	Set the Read Only property to TRUE

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 Call Signature Capture View
      ```

   c. Right-click the view you located in Step b, and then click Copy Record.
   d. Add an _PUI suffix to the name. For example:

      ```
      LS Pharma Call Signature Capture View_PUI
      ```

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 Call Signature Form Applet
      ```

   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_PUI

   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 Professional Call Form Applet
      ```

   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 NamePUI	LS Pharma Call Signature Form Applet_PUI

   e. In the Object Explorer, click Drilldown Object.
   f. In the Drilldown Objects list, query the Name property for Signature Capture Drilldown.
   g. Create a copy of this record, add the new drilldown to the record copy, and update the following field:

Name	New Value
Signature Capture DrillDownPUI	LS Pharma Call Signature Capture View_PUI

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 Calls 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. Create a copy of the LS Pharma Call Signature Capture View, and update the following field:

Old Value	New Value
LS Pharma Call Signature Capture View	LS Pharma Call Signature Capture View_PUI

10. Compile your modifications.
11. Administer 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_PUI

f. In the Responsibilities list, add a responsibility. Use the same field values that you noted in Step c.
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

Field	Value

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. Add the applet LS Pharma Call Signature Form Applet_PUI to the manifest administration as follows:

a. Log in to the client with administrative privileges.
b. Navigate to the Administration - Application screen, and then the Manifest Administration view.
c. Under UI Objects, create a new record with the following values:

Interactive Flag	Type	Usage Type	Name
N	Applet	Physical Renderer	LS Pharma Call Signature Form Applet_PUI

d. Under Object Expression, add the following child applet for the record created in Step c.

Interactive Flag	Expression	Level
N	Desktop	1

e. Under Files, set the following file values:

Interactive Flag	Name
N	3rdParty/jquery.signaturepad.min.js

13. Test your modifications.
    a. Log in to the Siebel Open UI client (for example, Siebel Pharma application).
    b. Navigate to a contact call where you want to capture the signature.
    c. Click Sign to open the Signature Capture view.
    d. Verify that the Signature Capture view applet displays correctly - that is, according to the customizations detailed in this procedure.

Customizing Applets to Capture Signatures for Siebel Mobile Applications

A *signature capture* is an electronic capture of a user signature. This topic describes how to customize applets to capture signatures in Siebel Mobile applications.

> **Note:** This task uses Siebel Pharma as an example, but the procedure is similar when modifying a different application. For more information about migrating signatures from High Interactivity to Siebel Open UI, see the topic about configuring the digital migration service for signatures and the topic about rendering signatures in the user interface in *Siebel Life Sciences Guide* .

To customize applets to capture signatures for Siebel Mobile applications

1. Create a new business component and add a new field.
    a. Create a new Signature business component with the values shown in the following table.

Property	Value
Name	Signature BC
Class	CSSBCBase

    b. Create a new Signature business component field with the values shown in the following table.

Property	Value
Name	Signature
Type	DTYPE_NOTE
Text Length	16,383
Force Activation	Selected

Property	Value

2. Create a new Form Applet with the values shown in the following table, adding an _PUI suffix to the name.

Name	Class	Business Component
Signature Form Applet_PUI	CSSFrameBase	Signature BC

3. 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 as required.

Name	Value
CanInvokeMethod: ClearSignature	TRUE
Parent BC Name, for example: - For Siebel Pharma, Parent BC Name is: Parent BC Name: Pharma Professional Call - Mobile - For Siebel Service, Parent BC Name is: Parent BC Name: Action	For example: - Pharma Professional Call - Mobile - Action
Signature Field	Signature
Signature Length	1600
Signature Min Length	5
Use Apply Drilldown	Y
Use Cancel Drilldown	Y

4. 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. **Note:** You can create other fields such as Contact First Name in addition to the Address field as required.
Signature Capture	Set the following properties: - Set the Field property to Signature. - Set the HTML Type property to InkData.
Apply Signature	Set the MethodInvoked property to ApplySignature.
Cancel Signature	Set the MethodInvoked property to CancelSignature.

5. Add an applet Web template:
    a. In the Object Explorer, click Applet Web Template.
    b. In the Applet Web Templates list, right-click and select new record.
    c. Set the following properties.

Property	Value
Name	Edit
Type	Edit
Web Template	SIA Applet Form Grid Layout - No Menu_OUI

6. Create the following new drilldown objects:
    a. In the Object Explorer, click Drilldown Object.
    b. In the Drilldown Objects list, configure the values shown in the following table as required for the Apply Drilldown and the Cancel Drilldown drilldown objects.

    > **Note:** The values shown in the following table (for View, Business Component, and so on) are examples only - you can choose a different view and business component as required.

Name	Hyperlink Field	View	Source Field	Business Component	Destination Field
Apply Drilldown	Address	LS Pharma Professional Call Execute View - Mobile	Activity Id	Pharma Professional Call - Mobile	Id
Cancel Drilldown	Address	LS Pharma Professional Call Execute View - Mobile	Activity Id	Pharma Professional Call - Mobile	Id

7. Expose the Controls in the Applet Web Template item as follows:
    a. In the Object Explorer, click Applet.
    b. Select Applet "Signature Form Applet_PUI", then right-click and select Edit Web Layout.
    c. Select Edit mode.
    d. Select and move the Signature field and the Apply Signature, Cancel Signature, and Clear Signature buttons on the Web Layout.
8. Compile your modifications.
9. Add the applet Signature Form Applet_PUI to the manifest administration as follows:
    a. Log in to the client with administrative privileges.
    b. Navigate to the Administration - Application screen, and then the Manifest Administration view.
    c. Under UI Objects, create a new record with the following values:

Interactive Flag	Type	Usage Type	Name
N	Applet	Physical Renderer	Signature Form Applet_PUI

    d. Under Object Expression, add the following child applet for the record created in Step c.

Interactive Flag	Expression	Level
N	*<Empty>*	2
N	Mobile	1

Interactive Flag	Expression	Level

e. Under Files, set the following file values:

Interactive Flag	Name
N	3rdParty/jquery.signaturepad.min.js
N	siebel/signviewpr.js

f. Under UI Objects, create a new record with the following values:

Interactive Flag	Type	Usage Type	Name
N	Applet	Presentation Model	Signature Form Applet_PUI

g. Under Object Expression, add the following child applet for the record created in step f.

Interactive Flag	Expression	Level
N	Mobile	1

h. Under Files, set the following file values:

Interactive Flag	Name
N	siebel/signviewpm.js

10. Test your modifications.

a. Log in to the Siebel Open UI client.
b. Navigate to a view where the Signature Form Applet_PUI is exposed.
c. Verify that the Signature Capture view applet displays correctly - that is, according to the customizations detailed in this procedure.

Customizing Applets to Display Record Counts for Navigation Links

This topic describes how to customize an applet to display record counts for navigation links. The Navigation Links Runtime business component includes fields applicable to record counts for navigation links. Also, the CCAppletList_tile_NavLink_ss Web template includes the siebui-record-count class to hold the record count and the error image that appears when the record count cannot be retrieved.

After you complete the procedure in this topic, administrators can access the Siebel Open UI client, and use the Navigation Links view of the Administration - Application screen to display the number of records in the location to which a link navigates. For navigation links that appear in tiles, this record count appears beneath the display name for the navigation link. For navigation links that appear in lists, this record count appears in the Record Count field in the lists.

Currently, record counts for navigation links are available for only the Siebel eService application.

To customize an applet to display record counts for navigation links

1. Configure the manifest:
    a. Log in to a Siebel client with administrative privileges.

       For more information about the screens that you use in this step, see *Configuring Manifests*.
    b. Navigate to the Administration - Application screen, and then the Manifest Files view.
    c. In the Files list, add the following file if it does not already exist in the list:

       ```
       siebel/navlinkpr.js
       ```
    d. Navigate to the Administration - Application screen, and then the Manifest Administration view.
    e. In the UI Objects list, select the applet for which you want to display record counts for navigation links.

       The applet must have a value in the Usage Type field of Physical Renderer.
    f. In the Object Expression list, select the appropriate expression for the applet.
    g. In the Files list, add the following file:

       ```
       siebel/navlinkpr.js
       ```
2. If navigation links appear in a list, then include the Record Count field in the list:
    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 in which to include the Record Count field.

       This applet has a Business Component property of Navigation Links Runtime.
    d. In the Object Explorer, expand the Applet tree, then expand the List tree, and then click List Column.
    e. In the List Columns list, create a new list column with a value of List Record Count for the Field property.
    f. Compile your work.

3. (Optional) Change the location of the record count number from its default location in the corner of tiles:
   a. Open the appropriate CSS file for editing.

      For example, open the theme-aurora.css file, which is located at `AI_Install_Dir\applicationcontainer_external\siebelwebroot\files`.
   b. Locate the siebui-record-count class in this file.
   c. Change the information for this class to change the location of the record count number.

      For example, to center the record count number in the center of tiles, change the information for this class as follows:

```
siebel-record-count {
 text-align: center;
 padding-top: 10px;
}
```

Customizing Applets for Homepage Views in Redwood Theme

To convert the homepage applet, you can modify the web template that the applet references.

To modify the web template to convert homepage applets with record count

1. In Siebel Web Tools, navigate to **Siebel Object** > **Applet** and query for the applet name.
2. Go to **Applet** > **Control** and then create two new controls with the following attributes.

   Clone `AppletTitle` and change the attributes as mentioned below:

Field	Value
Name	AppletTitleAsLabel
Caption - String Reference	<Provide the appropriate Symbolic String>
HTML Type	Label

Field	Value
Name	ViewAllLink
Caption - String Reference	SBL_APP_HOMEPAGE_VIEWALL
HTML Type	Link

Field	Value
Method Invoked	GotoView

3. For the ViewAllLink control, expand the Control object, select the User Props
   a. Create a new record
   b. Enter the name as View
   c. Enter value as <Enter the Destination View name where it should navigate to when the link is clicked>
4. Go to **Applet Web Template**.
5. Clone the **Tile** Web Template.

   Modify the Name, Sequence, Type, and name of the newly cloned Applet Web Template as follows:

Field	Value
Name	Redwood
Sequence	1001
Type	Base
Web Template	`CCAppletListHomePageTile`(Redwood Theme)

> **Note:** If there is no sequence number for the **Tile** Web Template, you must assign a Sequence Number. For example:**Sequence**: 1

6. Go to the **Applet Web Template Item** and map the following to the web template:
   - Add the two new controls created in Step 2 and provide the appropriate Item Identifier.

     Also, add the record count control and provide the appropriate Item Identifier.
   - Add any three List Columns that you wish to see in the applet and provide the appropriate Item Identifier.

   a. Control mapping to the Redwood Web Template.

Field	Value
Name	AppletTitle
Control	AppletTitleAsLabel
Item Identifier	185

Field	Value
Name	ViewAll
Control	ViewAllLink
Item Identifier	184

b. List columns mapping to the Redwood web template (Data Rendering).

Field	Value
Name	<Data Field>
Control	<Choose the appropriate list column>
Item Identifier	510

Field	Value
Name	<Link Field>
Control	<Choose the appropriate list column that has drill down>
Item Identifier	511

Field	Value
Name	<Data Field>
Control	<Choose the appropriate list column>
Item Identifier	512

Field	Value

Note: `Item Identifier = 511` is reserved as a drill-down field, so it is advisable to have a list column where drill-down is configured.

7. After making the Redwood Web Template changes in the Web Tools and deliver it, create a new manifest entry for the applet to conditionally select the new Web Template only when the Redwood theme is applied.

Manifest Administration

1. Navigate to **Manifest Administration** View.
2. In the UI Objects list, specify the following applet.

Field	Value
Type	Applet
Usage Type	Physical Renderer
Name	<Applet Name>

3. In the Object Expression list, add the following expression.

Field	Value
Expression	Redwood Theme
Level	1

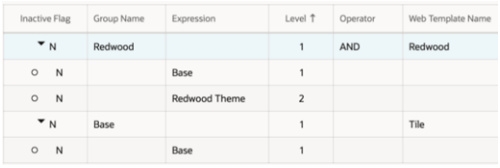
4. In the Files list, add the following file:

```
siebel/TileLayoutPR.js
```

5. In the Object Expression list, add expressions until this list resembles the configuration shown in the following table and image.

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
N	Redwood	<empty>	1	AND	Redwood
N	<empty>	Base	1	<empty>	<empty>

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
N	<empty>	Redwood Theme	2	<empty>	<empty>
N	Base	<empty>	1	<empty>	Tile
N	<empty>	Base	1	<empty>	<empty>



Inactive Flag	Group Name	Expression	Level ↑	Operator	Web Template Name
N	Redwood		1	AND	Redwood
N		Base	1		
N		Redwood Theme	2		
N	Base		1		Tile
N		Base	1		

To modify the web template to convert homepage list applets

1. Identify the web template that the applet is using. Let us consider that the applet web template is of type **`Base/ Edit List`**
2. Navigate to **Siebel Object** > **Applet** and query for the Applet name.
3. Go to **Applet** > **Control** and create two new controls with the following attributes.

   Clone **`AppletTitle`** and change the attributes as mentioned below:

Field	Value
Name	AppletTitleAsLabel
Caption - String Reference	<Provide the appropriate Symbolic String>
HTML Type	Label

Field	Value
Name	ViewAll
Caption - String Reference	SBL_APP_HOMEPAGE_VIEWALL
HTML Type	Link
Method Invoked	GotoView

4. For the ViewAllLink control, expand the Control.
   a. Create a new record
   b. Enter the name as **View**
   c. Enter value as <Enter the Destination View name where it should navigate to when the link is clicked>
5. Navigate to **Applet Web Template**
   - Create a new record by cloning the `Edit List` Web Template.

Field	Value
Name	Redwood
Sequence	1001
Type	Edit List

> **Note:** If there is no sequence number for the **Tile** Web Template, you must assign a Sequence Number. For example:**Sequence**: 1

6. Go to **Applet Web Template Item** and map the following to the Web Template.
   - Map two new controls to the web template.
7. Control mapping to the Redwood web template

Field	Value
Name	AppletTitle
Control	AppletTitleAsLabel
Item Identifier	184

Field	Value
Name	ViewAll
Control	ViewAllLink
Item Identifier	90

8. After making the Redwood Web Template changes in the Web Tools and deliver it, create a new manifest entry for the applet to conditionally select the new web template only when the Redwood theme is applied.

Note: In the Applet Web Template, if all the available templates have the same `Type` , it is observed that the template is not selected dynamically, and the existing template is always chosen. For example, if there are three existing web templates and all are of `Type=Base`, and you add a new web template **Redwood**, it will not select the web template based on the expression provided in the Manifest Administration.

Note: The workaround here is to create a dummy Web Template just to include a second Type. For example, if all web templates are of `Type=Base`, one dummy web template should be of `Type=Query/EditList`.

Manifest Administration

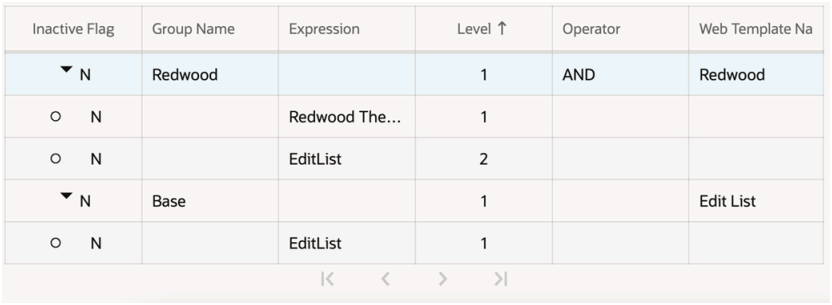
1. Navigate to **Manifest Administration** View.
2. In the UI Objects list, specify the following applet.

Field	Value
Type	Applet
Usage Type	Web Template
Name	<Applet Name>

3. In the Object Expression list, add expressions until this list resembles the configuration shown in the following table and image.

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
N	Redwood	<empty>	1	AND	Redwood
N	<empty>	Redwood Theme	1	<empty>	<empty>
N	<empty>	EditList	2	<empty>	<empty>
N	You can enter the name of the group that you are going to associate a Web Template Name	<empty>	1	<empty>	Use the value as either Base or Edit List based on the Applet Web Template that you are going to display in the application.
N	<empty>	Use the value as either Base or	1	<empty>	<empty>

Inactive Flag	Group Name	Expression	Level	Operator	Web Template Name
		EditList based on the Applet Web Template that you are going to display in the application.			



Customizing Controls

This topic describes how to customize a control. It includes the following information:

- *Creating and Managing Client-Side Controls*
- *Displaying Control Labels in Different Languages*
- *Customizing Presentation Models to Display Control Labels in Different Languages*
- *Customizing the Busy Cursor to Display While a Business Service Executes*
- *Creating Property Sets for Client-Side Controls*
- *Properties That You Can Specify for Client-Side Controls*
- *Text Copy of the Client Control Presentation Model File*

This book includes a number of other topics that also customize controls. For more information about:

- Overview information about customizing controls, see *Examples of How You Can Customize Siebel Open UI*, *Example Client Customizations*, and *Guidelines for Customizing Siebel Open UI*.
- Adding a control to a presentation model, see *Customizing the Setup Logic of the Presentation Model*.
- Modifying a list column control so that Siebel Open UI stores the value of the control check box, see *Customizing the Presentation Model to Identify the Records to Delete*.
- Customizing control user properties, see *Customizing Control User Properties for Presentation Models*.
- Accessing a proxy object for an active control, see *Accessing Proxy Objects*.
- Customizing control themes, see *Customizing Themes*.
- Rendering controls according to control type, see *Customizing List Applets to Render as Maps*.
- Adding a control that does a static drill-down, see *Adding Static Drilldowns to Applets*.
- Customizing controls in an applet, see *Customizing Applets to Capture Signatures from Desktop Applications*.

- Adding controls to the calendar, *Customizing a Resource Scheduler*.

Creating and Managing Client-Side Controls

The example in this topic describes how to create a text box that the Siebel Open UI client displays, and is not represented on the Siebel server. This is a Siebel Open UI client implementation, and as such, data will not be maintained after the user navigates away from the view containing this type of control. You can also create similar controls, such as date/time, check box, combobox, and so on.

This example shows how to configure client-side controls in list applets, however, the same principals can be applied to form applets.

To create controls in the client

1. Create a custom presentation model:

   a. Use a JavaScript editor to create a new file named clientctrlpmodel.js. Save this file in the following folder:

   ```
   siebel\custom
   ```

   For more information about:

   - The complete presentation model that this example uses, see *Text Copy of the Client Control Presentation Model File*.
   - This folder, see *Organizing Files That You Customize*.

   b. Add the following code to the file that you created in Step a.

   This code does the basic set up:

   ```
   if( typeof( SiebelAppFacade.ClientCtrlPModel ) === "undefined" ){
    SiebelJS.Namespace( 'SiebelAppFacade.ClientCtrlPModel' );
    //Module with its dependencies
    define("siebel/custom/clientctrlpmodel", [], function () {
    SiebelAppFacade.ClientCtrlPModel = ( function(){
    var consts = SiebelJS.Dependency( "SiebelApp.Constants" );
    /* *
    * Constructor Function - ClientCtrlPModel
    *
    * Parameter - Be a good citizen. Pass All parameter to superclass.
    * */
    function ClientCtrlPModel(proxy){
    var m_recordset = [];
    SiebelAppFacade.ClientCtrlPModel.superclass.constructor.call( this, proxy);
   ```

   c. Add the client control:

   ```
    this.AddMethod("AddClientControl", null, { core : true } );
    // add into method array
    this.GetClientRecordSet = function( ) {
    return m_recordset ;
    };
    }
   ```

   For more information, see *AddMethod Method* and *AddClientControl Method*.

d. Extend the ListPresentationModel object:

```
 /* Siebel OpenUI uses the ListPresentationModel object to initialize every
list applet. So, to maintain the functionality that ListPresentationModel
provides, you extend it.*/
 SiebelJS.Extend( ClientCtrlPModel, SiebelAppFacade.ListPresentationModel );
 ClientCtrlPModel.prototype.Init = function(){
 SiebelAppFacade.ClientCtrlPModel.superclass.Init.call( this );
```

e. Determine whether or not Siebel Open UI has removed the focus from the field in the applet, and then temporarily store the value that the user entered in the control:

```
 /* Attach Post Handler for LeaveField */
 this.AddMethod( "LeaveField", PostLeaveField, { sequence : false, scope :
this } );
```

For more information, see *LeaveField Method* and *PreGetFormattedFieldValue Method*.

f. Get the format that the field uses to store the value for the control:

```
 /* Attach Pre Handler for GetFormattedFieldValue */
 this.AddMethod("GetFormattedFieldValue", PreGetFormattedFieldValue, {
sequence : true, scope : this } );
 /* Attach Handler for Delete Record Notification as well */
 this.AttachNotificationHandler( consts.get(
"SWE_PROP_BC_NOTI_DELETE_RECORD" ), HandleDeleteNotification );
```

For more information, see *GetFormattedFieldValue Method*.

g. Get the data from memory stored in Step f, and then display this data in the client control:

```
function PreGetFormattedFieldValue(control, bUseWS, recIndex, returnStructure){
 if (utils.IsEmpty(recIndex)){
 recIndex = this.Get("GetSelection");
 }
 if (recIndex >=0) {
 var clientObj = this.GetClientRecordSet();
 var recordSet=this.Get("GetRawRecordSet");
 var id = recordSet[recIndex]["Id"];
 var flag = false;
 var value;
 switch(control.GetName()){
 case "TestEdit":
 value = recordSet[recIndex]["Name"];
 flag = true;
 break;
 }
 if (flag){
 if( clientObj[id] && clientObj[id][control.GetName()] ){
 value = clientObj[id][control.GetName()];
 }
 else if (clientObj[id]){
 clientObj[id ][control.GetName()] = value;
 }
 else{
 var recordclient = {};
 recordclient[control.GetName()] = value;
 clientObj[id] = recordclient;
 }
 returnStructure[ "CancelOperation" ] = true;
 returnStructure[ "ReturnValue" ] = value;
 }
 }
```

```
}
```

For more information, see *PreGetFormattedFieldValue Method*.

h. Save the value after the user leaves the client control:

```
function PostLeaveField( control, value, notLeave, returnStructure ){
 var clientObj = this.GetClientRecordSet();
 var currSel = this.Get( "GetSelection" );
 var recordSet=this.Get("GetRawRecordSet");
 var id = recordSet[currSel]["Id"];
 if (clientObj[id]){
 switch(control.GetName()){
 case "TestEdit":
 clientObj[id][control.GetName()] = returnStructure[ "ReturnValue" ] ;
 break;
 }
 }
}
```

For more information, see *PreGetFormattedFieldValue Method*.

i. Delete the reference to the record data that Siebel Open UI stored in the client for the control:

```
 function HandleDeleteNotification(propSet){
 var activeRow = propSet.GetProperty( consts.get(
"SWE_PROP_BC_NOTI_ACTIVE_ROW" ) );
 if( activeRow === this.Get( "GetSelection" ) ){
 var delObj = this.GetClientRecordSet();
 delObj[ activeRow ] = null;
 }
 }
```

For more information, see *HandleDeleteNotification Method*.

j. Create a property set for the control.

For this example, you use the following code to create a property set for the text box control:

```
 ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
 /// Specify the property set for Edit box
 SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
 var ctrlTxtInfo = SiebelAppFacade.PresentationModel.GetCtrlTemplate
("TestEdit", "Test Edit", consts.get( "SWE_CTRL_TEXTAREA" ), 1);
```

For more information about this code, see *Creating Property Sets for Client-Side Controls*.

k. Add the property set information so that Siebel Open UI can add it to the proxy:

```
this.ExecuteMethod( "AddClientControl", ctrlTxtInfo );
```

l. Return the ClientCtrlPModel that you set up in Step b:

```
 };
 return ClientCtrlPModel;
 } ());
 return "SiebelAppFacade.ClientCtrlPModel";
 });
}
```

2. Configure the manifest:

   a. Log in to a 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/clientctrlpmodel.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	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/clientctrlpmodel.js
```

h. To refresh the manifest, log out of the client, and then log back in to the client.

3. Test your work:

a. Navigate to any list applet, and then verify that it displays the control that you added.

In Step 1, Step b, you extended the ListPresentationModel object that Siebel Open UI uses to display every list applet. So, you can navigate to any list applet.

Creating Property Sets for Client- Side Controls

You can use the following code to create a property set for a control that Siebel Open UI displays in the client:

```
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
```

```
var variable_name= SiebelAppFacade.PresentationModel.GetCtrlTemplate
 ("control_name", "display_name", consts.get( "control_type" ), column_index );
 ctrlComboInfo.SetPropertyStr(consts.get("control_property"),
"property_attribute")
```

where:

- control_name, display_name, control_type, and column_index are arguments of the GetCtrlTemplate method. For more information about these arguments, see *GetCtrlTemplate Method*.
- control_property specifies a control property. For example, SWE_PROP_WIDTH specifies the width of the control, in pixels.
- property_attribute specifies an attribute of the control that control_property specifies. For example, 200 sets the width of the control to 200 pixels.

For example, the following code creates a variable named ctrlComboInfo for the TestCombo control. It sets the width and height of this control to 200 pixels, and centers it.

```
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
var ctrlComboInfo = SiebelAppFacade.PresentationModel.GetCtrlTemplate ("TestCombo",
 "Test Drop Down", consts.get( "SWE_CTRL_COMBOBOX" ), 10 );
 ctrlComboInfo.SetPropertyStr(consts.get("SWE_PROP_WIDTH"), "200")
 ctrlComboInfo.SetPropertyStr(consts.get("SWE_PROP_HEIGHT"), "200")
 ctrlChkboxInfo.SetPropertyStr(consts.get("SWE_PROP_JUSTIFICATION"), "center");
```

For more information about control_property and property_attribute, see *Properties That You Can Specify for Client-Side Controls*. For more information about other control properties that you can specify, such as Sort or Vertical Scroll, see the topic that describes the control Applet Object Type in *Siebel Object Types Reference* .

Properties That You Can Specify for Client-Side Controls

The following table describes the properties that you can specify for controls. The Comparable Applet Control or Description column of this table includes the name of the applet control property that is similar to the SWE control property. If no applet control property is similar to the SWE control property, then this column includes a description. For more information about these applet control properties, see the topic that describes controls in the applet object types section of *Siebel Object Types Reference* .

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_CURR_FLD	Identifies the field that is currently chosen.
SWE_PROP_CASE_SENSITIVE	Specifies to make text in the control case-sensitive.
SWE_PROP_DISP_FORMAT	Display Format
SWE_PROP_DISP_MODE	HTML Display Mode
SWE_PROP_DISP_MAX_CHARS	HTML Max Chars Displayed
SWE_PROP_DISP_NAME	Specifies the label that Siebel Open UI uses to identify this control in the client.

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_FLD_NAME	Field Name
SWE_PROP_HEIGHT	HTML Height
SWE_PROP_HTML_ATTRIBUTE	HTML Attributes
SWE_PROP_IS_BOUND_PICK	Specifies that the control is a bound picklist.
SWE_PROP_IS_ENCODE	HTML Display Mode
SWE_PROP_INPUTMETHOD	MethodInvoked
SWE_PROP_JUSTIFICATION	Text Alignment
SWE_PROP_LABEL_JUSTIFICATION	Specifies the text alignment for a column header that Siebel Open UI displays in a list control.
SWE_PROP_MAX_SIZE	HTML Max Chars Displayed
SWE_PROP_NAME	Name
SWE_PROP_PICK_APLT	Pick Applet
SWE_PROP_POPUP_HEIGHT	Specifies the height of the popup dialog box, in pixels.
SWE_PROP_PROMPT	Prompt Text
SWE_PROP_POPUP_WIDTH	Specifies the width of the popup dialog box, in pixels.
SWE_PROP_IS_DYNAMIC	Specifies whether or not Siebel Open UI dynamically displays values in the control.
SWE_PROP_SPAN	Specifies to span control contents across multiple fields. This property is not applicable for list controls.
SWE_PROP_SEQ	HTML Sequence
SWE_PROP_TYPE	Type, HTML Type, or Field Retrieval Type
SWE_PROP_WIDTH	Width
SWE_PROP_COLINDEX	Specifies the index number of a column.
SWE_PROP_ICON_MAP	Bitmap

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_IS_SORTABLE	Sort

Text Copy of the Client Control Presentation Model File

The following code from the clientctrlpmodel.js file adds example controls to the client. You can examine this code for your reference. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support:

```
if( typeof( SiebelAppFacade.ClientCtrlPModel ) === "undefined" ){
 SiebelJS.Namespace( 'SiebelAppFacade.ClientCtrlPModel' );
 //Module with its dependencies
 define("siebel/custom/clientctrlpmodel", [], function () {
 SiebelAppFacade.ClientCtrlPModel = ( function(){
 var consts = SiebelJS.Dependency( "SiebelApp.Constants" );
 /* *
 * Constructor Function - ClientCtrlPModel
 *
 * Parameter - Be a good citizen. Pass All parameter to superclass.
 * */
 function ClientCtrlPModel(proxy){
 var m_recordset = [];
 SiebelAppFacade.ClientCtrlPModel.superclass.constructor.call( this, proxy);
 this.AddMethod( "AddClientControl", null, { core : true } );
 // add into method array
 this.GetClientRecordSet = function( ) {
 return m_recordset ;
 };
 }
 /* Siebel OpenUI uses the ListPresentationModel object to initialize every list
applet. So, to maintain the functionality that ListPresentationModel provides, you
extend it.*/
 SiebelJS.Extend( ClientCtrlPModel, SiebelAppFacade.ListPresentationModel );
 ClientCtrlPModel.prototype.Init = function(){
 SiebelAppFacade.ClientCtrlPModel.superclass.Init.call( this );
 /* Attach Post Handler for LeaveField */
 this.AddMethod( "LeaveField", PostLeaveField, { sequence : false, scope : this
} );
 /* Attach Pre Handler for GetFormattedFieldValue */
 this.AddMethod("GetFormattedFieldValue", PreGetFormattedFieldValue, { sequence
: true, scope : this } );
 /* Attach Handler for Delete Record Notification as well */
 this.AttachNotificationHandler( consts.get( "SWE_PROP_BC_NOTI_DELETE_RECORD"
), HandleDeleteNotification );
 };
 function PreGetFormattedFieldValue(control, bUseWS, recIndex, returnStructure){
 if (utils.IsEmpty(recIndex)){
 recIndex = this.Get("GetSelection");
 }
 if (recIndex >=0) {
 var clientObj = this.GetClientRecordSet();
 var recordSet=this.Get("GetRawRecordSet");
 var id = recordSet[recIndex]["Id"];
 var flag = false;
 var value;
 switch(control.GetName()){
 case "TestEdit":
 value = recordSet[recIndex]["Name"];
 flag = true;
 break;
 }
```

```
if (flag){
if( clientObj[id] && clientObj[id][control.GetName()] ){
value = clientObj[id][control.GetName()];
}
else if (clientObj[id]){
clientObj[id ][control.GetName()] = value;
}
else{
var recordclient = {};
recordclient[control.GetName()] = value;
clientObj[id] = recordclient;
}
returnStructure[ "CancelOperation" ] = true;
returnStructure[ "ReturnValue" ] = value;
}
}
}
function PostLeaveField( control, value, notLeave, returnStructure ){
var clientObj = this.GetClientRecordSet();
var currSel = this.Get( "GetSelection" );
var recordSet=this.Get("GetRawRecordSet");
var id = recordSet[currSel]["Id"];
if (clientObj[id]){
switch(control.GetName()){
case "TestEdit":
clientObj[id][control.GetName()] = returnStructure[ "ReturnValue" ] ;
break;
}
}
}
function HandleDeleteNotification(propSet){
var activeRow = propSet.GetProperty( consts.get(
"SWE_PROP_BC_NOTI_ACTIVE_ROW" ) );
if( activeRow === this.Get( "GetSelection" ) ){
var delObj = this.GetClientRecordSet();
delObj[ activeRow ] = null;
}
}
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// PS Attribute info for Edit box
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
var ctrlTxtInfo = SiebelAppFacade.PresentationModel.GetCtrlTemplate
("TestEdit", "Test Edit", consts.get( "SWE_CTRL_TEXTAREA" ), 1);
this.ExecuteMethod( "AddClientControl", ctrlTxtInfo );
};
return ClientCtrlPModel;
} ());
return "SiebelAppFacade.ClientCtrlPModel";
});
}
```

Configuring Client-Side Multi-Select

Siebel Open UI uses a client-side control implementation to display a Multi-Select check box column in list applets. While this is primarily intended for touch-based devices where multiple selection of rows is not possible using the Shift + Click or Ctrl + Click, it can also be configured for desktop browsers.

The Multi Row Select Checkbox Display user property controls the behavior and availability of the client-side multi-select check boxes. The property can have the following values:

- **TOUCH-HIDE.** The multi-select column does not appear on touch devices.
- **TOUCH-SHOW.** The multi-select column appears on touch devices.
- **NONTOUCH-HIDE.** The multi-select column does not appear on desktop and non-touch based devices.

- **NONTOUCH-SHOW.** The multi-select column appears on desktops and non-touch based Touch devices.

When the user property is not configured for an applet, the default behavior is to show the Multi-Select column on touch devices and hide the column on non-touch devices. Administrators can use the user property to override this behavior on a per-list applet basis.

To configure a multi-select check box for a list applet

1. Open Siebel Tools.

   For more information, see *Using Siebel Tools* .
2. In the Object Explorer, click Applet.
3. In the Applets list, locate the applet that you want to configure.
4. Add the applet user property to the applet that you located in Step 3:
   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 property with one of the possible values:

Name	Values
Multi Row Select Checkbox Display	TOUCH-HIDE, TOUCH-SHOW, NONTOUCH-HIDE, NONTOUCH-SHOW

5. Compile the applet object.
6. Restart the Siebel server.

   Your changes will now be visible in the Siebel Open UI client.

Displaying Control Labels in Different Languages

This topic describes how to modify the custom_messages.js file so that Siebel Open UI displays the text for a client-side control label according to the language that the client browser uses. You can also modify the presentation model instead of modifying the custom_messages.js file. For more information about how to do this, see *Customizing Presentation Models to Display Control Labels in Different Languages*. For more information about language support, see *Languages That Siebel Open UI Supports*.

To display control labels in different languages

1. Create the following folder structure, if it does not exist, where <lang> is the language code such as DEU:

   ```
   AI_DIR/siebel/scripts/siebel/custom/<lang>
   ```

2. Copy custom_messages.js from the following folder to the folder created in Step 1:

   ```
   AI_DIR/siebel/scripts/siebel/samples
   ```

3. Open the file you saved in Step 2 using a JavaScript editor.
4. Locate the following code:

   ```
   function _SWEgetGlobalCustomMsgAry()
   {
    if (! _SWEbCMsgInit)
   ```

```
 {
 SWEbCMsgInit = true;
 }
 return _SWEcustommsgAry;
}
```

5. Add the code in bold to the code that you located in Step 4:

```
function _SWEgetGlobalCustomMsgAry()
{
 if (! _SWEbCMsgInit)
 {
 SWEbCMsgInit = true;
 SWEcustommsgAry["CUSTOM_ID"] = "custom_string";
 }
 return _SWEcustommsgAry;
}
```

   where:

   - CUSTOM_ID is a string that you use to reference the custom_string. You can use any value for CUSTOM_ID.
   - custom_string is a text string that includes text that you manually translate into the language that your deployment requires.

6. Add the following code at the beginning of the new file:

```
SiebelApp.S_App.LocaleObject.m_bClientStringsInitialized = false;
```

7. Save the file.
8. Navigate to the Administration - Application screen of your Siebel client, then the Manifest Expressions view.

   Note: You need to have administrative privileges in the Siebel client.

9. In the Expressions list, add the following expression for the language where the string will be used:

Field	Name	Expression
Value	<Name of the expression> such as Dutch	Language()= “<Language code>” such as “DEU”

10. Navigate to the Manifest Files view to add the new file under manifest files.
11. Add an entry for custom_message.js in the new folder created in Step 1 as follows:

```
siebel/custom/deu/custom_messages.js
```

12. Navigate to the Manifest Administration view to add a new record.
13. In the Objects UI list, create a new entry and specify the object as follows:

Field	Type	Usage Type	Name
Value	Application	Common	PLATFORM INDEPENDENT

14. In the Object Expression list, add the following subexpression:

Field	Group Name	Expression	Level
Value	Not Applicable	DEU	1

15. In the Files list, click Add.
16. In the Files dialog box, query for the path and filename that you added in Step 11.
17. Click Go.
18. Save the changes to the manifest.
19. Customized translated string is now available for creating client side control via customized presentation model. Use the locale object GetLocalString API to retrieve the customized string. Update customized presentation model and provide appropriate values to the GetCtrlTemplate API to retrieve the string.
20. Log out of the Siebel client, clear browser cache, and log in again.
21. Test your work:
    a. Navigate to the screen that includes the control that Siebel Open UI uses to display the translated string that you modified in Step 4.

Customizing Presentation Models to Display Control Labels in Different Languages

This topic describes how to customize a presentation model so that it displays a client-side control label in a different language instead of modifying the custom_messages.js file.

To customize presentation models to display control labels in different languages

1. Use a JavaScript editor to open the presentation model that Siebel Open UI uses to display the control label that you must modify.

   For more information, see *About the Presentation Model*.
2. Add the following code to call the ExecuteMethod method that the presentation model uses. You can add this code anywhere in the presentation model file:

   ```
   pm.ExecuteMethod("AddLocalString","ID","custom_string");
   ```

   where:

   AddLocalString is the name of the method that ExecuteMethod calls to add your custom string.

   For more information about how this example uses ID and custom_string, see *Displaying Control Labels in Different Languages*. For more information about these methods, see *AddLocalString Method* and *ExecuteMethod Method*.

   For example, add the following code:

   ```
   pm.ExecuteMethod("AddLocalString", "New", "Neu");
   pm.ExecuteMethod("AddLocalString", "Delete", "Löschen");
   ```
3. Test your work:

a. Navigate to the screen that includes the control that Siebel Open UI uses to display the translated string that you modified in Step 2.
b. Verify that the control displays the translated string.

Customizing the Busy Cursor to Display While a Business Service Executes

You can force a busy cursor to appear while a selected business service is executing. The example in this topic describes how to configure this behavior.

There is a system preference for Busy Cursor Timeout. For more information, see *About Preferences*.

To display a busy cursor while a Business Service executes

1. Create an applet with a button that invokes an always-on method.
2. Create a physical renderer to respond to the method invocation.
3. Create a business service to be invoked.
   a. The following is an example of a business service.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
var nReturn = ContinueOperation;
switch (MethodName) {
case "ExampleMethod":
var count;
for (var i=1; i<4000000; i++) {
Outputs.SetProperty("OutputProperty", "OutputValue");
Outputs.SetProperty("ReturnedProperty", Inputs.GetProperty("InputProperty"));
count++;
 }
Return = CancelOperation;
break;
}
 return (nReturn);
}
```

4. Make sure the business service is invokable from the client using the application user property.
5. Update the physical renderer to invoke the business service workOnBusyCursor control upon method invocation.
6. Test your work:
   a. Navigate to any list applet, and then verify that it displays the control that you added.

Creating Property Sets for Client-Side Controls

You can use the following code to create a property set for a control that Siebel Open UI displays in the client:

```
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
var variable_name= SiebelAppFacade.PresentationModel.GetCtrlTemplate
 ("control_name", "display_name", consts.get( "control_type" ), column_index );
```

```
 ctrlComboInfo.SetPropertyStr(consts.get("control_property"),
"property_attribute")
```

where:

- variable_name specifies the name of a variable.
- control_name, display_name, control_type, and column_index are arguments of the GetCtrlTemplate method. For more information about these arguments, see *GetCtrlTemplate Method*.
- control_property specifies a control property. For example, SWE_PROP_WIDTH specifies the width of the control, in pixels.
- property_attribute specifies an attribute of the control that control_property specifies. For example, for the SWE_PROP_WIDTH property, a value of 200 sets the width of the control to 200 pixels.

For example, the following code creates a variable named ctrlComboInfo for the TestCombo control. It sets the width and height of this control to 200 pixels, and centers it

```
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
/// Specify the property set for the control
SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
var ctrlComboInfo = SiebelAppFacade.PresentationModel.GetCtrlTemplate ("TestCombo",
 "Test Drop Down", consts.get( "SWE_CTRL_COMBOBOX" ), 10 );
 ctrlComboInfo.SetPropertyStr(consts.get("SWE_PROP_WIDTH"), "200")
 ctrlComboInfo.SetPropertyStr(consts.get("SWE_PROP_HEIGHT"), "200")
 ctrlChkboxInfo.SetPropertyStr(consts.get("SWE_PROP_JUSTIFICATION"), "center");
```

For more information about control_property and property_attribute, see *Properties That You Can Specify for Client-Side Controls*. For more information about other control properties that you can specify, such as Sort or Vertical Scroll, see the topic that describes the control Applet Object Type in *Siebel Object Types Reference* .

Properties That You Can Specify for Client-Side Controls

The following table describes the properties that you can specify for controls. The Comparable Applet Control or Description column of this table includes the name of the applet control property that is similar to the SWE control property. If no applet control property is similar to the SWE control property, then this column includes a description. For more information about these applet control properties, see the topic that describes controls in the applet object types section of *Siebel Object Types Reference* .

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_CURR_FLD	Specifies the field that is currently chosen.
SWE_PROP_CASE_SENSITIVE	Specifies to make text in the control case-sensitive.
SWE_PROP_DISP_FORMAT	Display Format
SWE_PROP_DISP_MODE	HTML Display Mode
SWE_PROP_DISP_MAX_CHARS	HTML Max Chars Displayed

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_DISP_NAME	Specifies the label that Siebel Open UI uses to identify this control in the client.
SWE_PROP_FLD_NAME	Field Name
SWE_PROP_HEIGHT	HTML Height
SWE_PROP_HTML_ATTRIBUTE	HTML Attributes
SWE_PROP_IS_BOUND_PICK	Specifies that the control is a bound picklist.
SWE_PROP_IS_ENCODE	HTML Display Mode
SWE_PROP_INPUTMETHOD	MethodInvoked
SWE_PROP_JUSTIFICATION	Text Alignment
SWE_PROP_LABEL_JUSTIFICATION	Specifies the text alignment for a column header that Siebel Open UI displays in a list control.
SWE_PROP_MAX_SIZE	HTML Max Chars Displayed
SWE_PROP_NAME	Name
SWE_PROP_PICK_APLT	Pick Applet
SWE_PROP_POPUP_HEIGHT	Specifies the height of the popup dialog box, in pixels.
SWE_PROP_PROMPT	Prompt Text
SWE_PROP_POPUP_WIDTH	Specifies the width of the popup dialog box, in pixels.
SWE_PROP_IS_DYNAMIC	Specifies whether or not Siebel Open UI dynamically displays values in the control.
SWE_PROP_SPAN	Specifies to span control contents across multiple fields. This property is not applicable for list controls.
SWE_PROP_SEQ	HTML Sequence
SWE_PROP_TYPE	Type, HTML Type, or Field Retrieval Type
SWE_PROP_WIDTH	Width
SWE_PROP_COLINDEX	Specifies the index number of a column.

SWE Control Property	Comparable Applet Control or Description
SWE_PROP_ICON_MAP	Bitmap
SWE_PROP_IS_SORTABLE	Sort

Text Copy of the Client Control Presentation Model File

The following code from the clientctrlpmodel.js file adds example controls to the client. You can examine this code for your reference. To get a copy of this file, see Article ID 1494998.1 on My Oracle Support:

```
if( typeof( SiebelAppFacade.ClientCtrlPModel ) === "undefined" ){
 SiebelJS.Namespace( 'SiebelAppFacade.ClientCtrlPModel' );
 //Module with its dependencies
 define("siebel/custom/clientctrlpmodel", [], function () {
 SiebelAppFacade.ClientCtrlPModel = ( function(){
 var consts = SiebelJS.Dependency( "SiebelApp.Constants" );
 /* *
 * Constructor Function - ClientCtrlPModel
 *
 * Parameter - Be a good citizen. Pass All parameter to superclass.
 * */
 function ClientCtrlPModel(proxy){
 var m_recordset = [];
 SiebelAppFacade.ClientCtrlPModel.superclass.constructor.call( this, proxy);
 this.AddMethod( "AddClientControl", null, { core : true } );
 // add into method array
 this.GetClientRecordSet = function( ) {
 return m_recordset ;
 };
 }
 /* Siebel OpenUI uses the ListPresentationModel object to initialize every list
applet. So, to maintain the functionality that ListPresentationModel provides, you
extend it.*/
 SiebelJS.Extend( ClientCtrlPModel, SiebelAppFacade.ListPresentationModel );
 ClientCtrlPModel.prototype.Init = function(){
 SiebelAppFacade.ClientCtrlPModel.superclass.Init.call( this );
 /* Attach Post Handler for LeaveField */
 this.AddMethod( "LeaveField", PostLeaveField, { sequence : false, scope : this
} );
 /* Attach Pre Handler for GetFormattedFieldValue */
 this.AddMethod("GetFormattedFieldValue", PreGetFormattedFieldValue, { sequence
: true, scope : this } );
 /* Attach Handler for Delete Record Notification as well */
 this.AttachNotificationHandler( consts.get( "SWE_PROP_BC_NOTI_DELETE_RECORD"
), HandleDeleteNotification );
 };
 function PreGetFormattedFieldValue(control, bUseWS, recIndex, returnStructure){
 if (utils.IsEmpty(recIndex)){
 recIndex = this.Get("GetSelection");
 }
 if (recIndex >=0) {
 var clientObj = this.GetClientRecordSet();
 var recordSet=this.Get("GetRawRecordSet");
 var id = recordSet[recIndex]["Id"];
 var flag = false;
 var value;
```

```
 switch(control.GetName()){
 case "TestEdit":
 value = recordSet[recIndex]["Name"];
 flag = true;
 break;
 }
 if (flag){
 if( clientObj[id] && clientObj[id][control.GetName()] ){
 value = clientObj[id][control.GetName()];
 }
 else if (clientObj[id]){
 clientObj[id ][control.GetName()] = value;
 }
 else{
 var recordclient = {};
 recordclient[control.GetName()] = value;
 clientObj[id] = recordclient;
 }
 returnStructure[ "CancelOperation" ] = true;
 returnStructure[ "ReturnValue" ] = value;
 }
 }
 }
 function PostLeaveField( control, value, notLeave, returnStructure ){
 var clientObj = this.GetClientRecordSet();
 var currSel = this.Get( "GetSelection" );
 var recordSet=this.Get("GetRawRecordSet");
 var id = recordSet[currSel]["Id"];
 if (clientObj[id]){
 switch(control.GetName()){
 case "TestEdit":
 clientObj[id][control.GetName()] = returnStructure[ "ReturnValue" ] ;
 break;
 }
 }
 }
 function HandleDeleteNotification(propSet){
 var activeRow = propSet.GetProperty( consts.get(
"SWE_PROP_BC_NOTI_ACTIVE_ROW" ) );
 if( activeRow === this.Get( "GetSelection" ) ){
 var delObj = this.GetClientRecordSet();
 delObj[ activeRow ] = null;
 }
 }
 ClientCtrlPModel.prototype.UpdateModel = function(psInfo){
 /// PS Attribute info for Edit box
 SiebelAppFacade.ClientCtrlPModel.superclass.UpdateModel.call( this, psInfo );
 var ctrlTxtInfo = SiebelAppFacade.PresentationModel.GetCtrlTemplate
("TestEdit", "Test Edit", consts.get( "SWE_CTRL_TEXTAREA" ), 1);
 this.ExecuteMethod( "AddClientControl", ctrlTxtInfo );
 };
 return ClientCtrlPModel;
 } ());
 return "SiebelAppFacade.ClientCtrlPModel";
 });
}
```

6 Customizing Calendars and Schedulers

Customizing Calendars and Schedulers

This chapter describes how to customize calendars and schedulers. It includes the following topics:

- *Customizing Calendars*
- *Customizing Resource Schedulers*

Customizing Calendars

This topic includes examples of customizing the calendar that Siebel Open UI displays. It includes the following information:

- *Using Fields to Customize Event Styles for the Calendar*
- *Allowing Users to Copy Items from List Applets to Create Calendar Events*
- *Customizing Event Styles for the Calendar*
- *Customizing Calendar Work Days*
- *Customizing How Calendars Display Timestamps*
- *Replacing Standard Interactivity Calendars*
- *Customizing How Users View Calendar Availability*
- *Customizing the Calendar All Day Slot*

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 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 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*.
10. Save the theme-calendar.css file.
11. Clear the browser cache.
12. Navigate to the Calendar view.
13. Make sure Siebel Open UI displays the correct styles.

Allowing Users to Copy Items from List Applets to Create Calendar Events

You can configure Siebel Open UI so that the user can copy an item from a list applet, and then paste it on a calendar to create an event.

Allowing users to copy items from list applets to create calendar events

1. Do Step 1 in the topic *Customizing Themes*.
2. Test your work:
   a. Log in to the client, and then navigate to the list applet that you modified in Step 1.
   b. Confirm that you can copy a record from the list applet, and then paste it on the calendar to create an event.

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, locate the following code:

   ```
   #color_square_LOV_name {color: custom_attributes important;}
   .fc-event-skin.calendar-EventStyle-LOV_name{
   {custom_attributes}
   ```

   where:

   - LOV_name identifies the event status that you must modify, such as Done or NotStarted.

     Note: The LOV name specified in the code should not include spaces.

   - custom_attributes specify the style properties you can modify, such as the background color or font type.
3. Modify the code that you located in Step 2, 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 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.

Customizing Calendar Work Days

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

Customizing How Calendars Display Timestamps

You specify an applet user property to customize how the calendar displays timestamps.

Note: If you have customized calendar to display timestamps, but still cannot see a timestamp, it might be hidden because the browser window is too small. In this case, modifications can be made to be made to the CSS.

To customize 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`
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`
8. Repeat Step 5 and Step 6 for the following applet user property:
   `Enable Monthly Time Display`

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
       ```

       > **Note:** This step describes how to search for specific standard interactivity calendar. If you want to replace a different standard interactivity calendar, query for it in this step.

    d. Modify the Class property from CSSSWEFrameCalGridLS to the following value:

       ```
       CSSSWEFrameActHICalGrid
       ```

       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 How Users View Calendar Availability

Calendar availability is the amount of free time in a user's agenda for a specific day. Available time is calculated by taking the number of working hours defined by the user, and subtracting any events already scheduled for that day within the working hours. You can configure Siebel Open UI to display the number of free hours available for a user in the monthly view. If you choose to show calendar availability, you will no longer see scheduled events in the monthly view. Instead, each day will have the available free hours displayed. This topic describes how to show and hide the calendar availability.

To show and hide calendar availability

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 BusyFreeTime
   ```

6. Set the Value property to one of the following values:
    - **Y.** Show calendar availability.
    - **N.** Hide calendar availability.

Customizing the Calendar All Day Slot

The calendar all day slot is an area in the calendar before the workday hours that lists all day events. All day events are calendar appointments that start and end at 00:00:00 in the user's time zone. By default, the all day slot is hidden. This topic describes how to show and hide the calendar all day slot.

To show or hide the calendar all day slot

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 AllDay Slot
   ```
6. Set the Value property to one of the following values:
    - **Y.** Show the calendar all day slot.
    - **N.** Hide the calendar all day slot.

Customizing Resource Schedulers

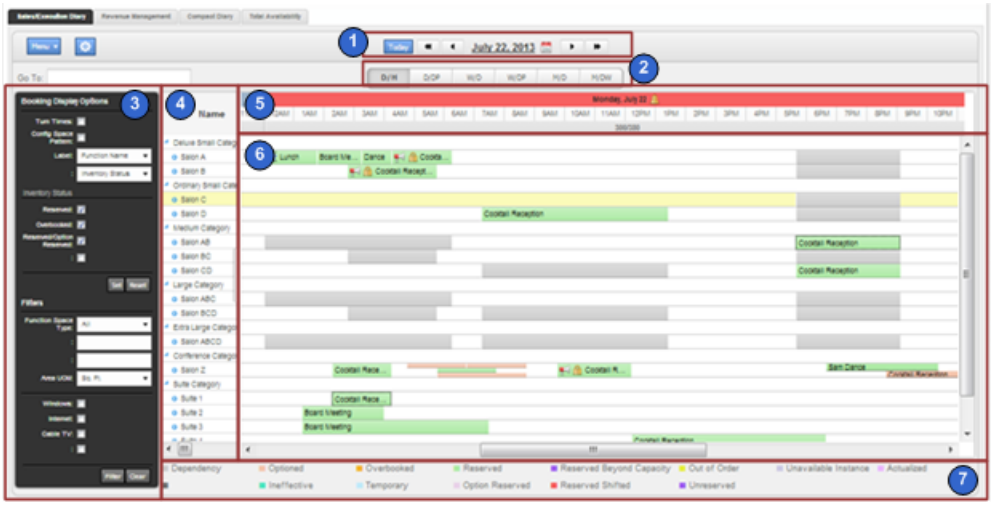
This topic describes how to customize a resource scheduler. It includes the following topics:

- *Overview of Customizing Resource Schedulers*
- *Customizing a Resource Scheduler*
- *Customizing the Filter Pane in Resource Schedulers*
- *Customizing the Resource Pane in Resource Schedulers*
- *Customizing the Timescale Pane in Resource Schedulers*
- *Customizing the Schedule Pane in Resource Schedulers*
- *Customizing Participant Availability in Resource Schedulers*
- *Customizing Tooltips in Resource Schedulers*

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

The following figure 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.



Explanation of Callouts

As shown in this image, 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.
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

In Siebel Open UI, 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*.

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, 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.
Date Padding for TimeScale LIC	time_scale_ identifier:number_of_pages	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*.
DateBar Navigation TS	time_scale_identifier:small_ date_change,big_date_change	Specify the date navigation buttons. For more information, see *Customizing the Date Navigation Buttons*.
Duration for TimeScale LIC	time_scale_identifier: number_of_days 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*.

Name	Value	Description
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.

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:
		`control_name,tag_name`
		where:
		- control_name specifies the name of the custom control.
		- tag_name specifies the tag name that you define in the Tag Name control user property in Step c in the topic *Customizing a Resource Scheduler*.
		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	

Name	Value	Description
System_Pref Field number For example, System_Pref Field 1, System_Pref Field 2, and so on.	"GntAXCtrl:Time Scale", "TST", "TNT_SHM_GNTAX_TIME_SCALE"	Specify the default values that Siebel Open UI uses in a resource scheduler. You can use the following abbreviations: - **TST.** Specifies the Time Scale. - **ST.** Specifies the start time of the schedule. - **ET.** Specifies the End time of the schedule. If a field is a LOV field, then you must specify the LOV name so that the code gets the language-dependent value. For more information about the abbreviations that the Value property contains, see *Using Abbreviations When Customizing the Resource Scheduler*.
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.
User_Pref Field number 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 select and move, 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.
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 select and move, 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 can use applet user properties to implement the optional customizations that your resource scheduler configuration requires. For more information about how to do this customization, see the following topics:

   - *Customizing the Filter Pane in Resource Schedulers*
   - *Customizing the Resource Pane in Resource Schedulers*
   - *Customizing the Timescale Pane in Resource Schedulers*
   - *Customizing the Schedule Pane in Resource Schedulers*
   - *Customizing Tooltips in Resource Schedulers*

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*.
   - 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, Step c to the layout.

    It is recommended that you position each of these controls on the other 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 number 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*.

    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 Siebel Open UI uses to reset the current date when the user clicks one of the following buttons:

- **Single arrow facing backwards.** Displays the previous date, small date change.
- **Single arrow facing forward.** Displays the next date, small date change.
- **Double arrow facing backwards.** Displays the previous date, large date change.
- **Double arrow facing forward.** Displays the next date, large date change.

Siebel Open UI displays these buttons at the start and to the end 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*.
   - 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 back arrow or the previous 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 arrow facing backwards or the double arrow facing forward.
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 back arrow, then Siebel Open UI displays August 8, 2013 as the current date.
  - The forward 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 arrow facing backwards, then Siebel Open UI displays July 15, 2013 as the current date.

- The double arrow facing forward, 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 back arrow, then Siebel Open UI displays August 14, 2013 as the current date.
  - The forward 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 arrow facing backwards button, then Siebel Open UI displays August 8, 2013 as the current date.
  - The double arrow facing forward, then Siebel Open UI displays August 22, 2013 as the current date.

Determining the Number That Siebel Open UI Uses to Identify Time

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 a 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 determines 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 in the topic *Customizing a Resource Scheduler*.
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 check box 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 4, Step e in the topic *Customizing a Resource Scheduler*.

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 in the topic *Customizing a Resource Scheduler*.
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 *number*	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.
Pane 0 Col *column number* Attr *column attribute number* 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: ○ **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. 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. 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.

Name	Value	Description
		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.
Pane 0 Col 0 Attr 3	DDFLD,Room Id	Specify the following items: ○ **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. 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.
Pane 0 Field *number*	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	Specify the view mode that this Resource pane supports. You must use the following numbers to indicate each view mode: ○ **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.

Name	Value	Description
		○ **9.** VIEW_SUBORG. You can use a comma to specify more than one view mode, where the comma separates each number. For example, 1,2,3.

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 *number*	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 Required,siebui-backup required`

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 in the topic *Customizing a Resource Scheduler*.
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.

Name	Value	Description
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.
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 number	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_GNTAX _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 number where number 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.

Name	Value	Description
Pane 1 BottomAxisBC Search Spec	[Product Type] = LookupValue (PRODUCT_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 3, Step b in the topic *Customizing a Resource Scheduler*.
   b. Add each of the following applet user properties, as required.

Name	Value	Description
Pane 1 Daypart number where number is the day part number.	Morning,NM,06:00:00, ST, 12:00:00,ET,21600,DUR	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 number	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.

Name	Value	Description
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. 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: o **Y.** Get the time scale color from the application object. o **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 number	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 *number* 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: ○ **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: ○ **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	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: o **0.** Sunday. o **1.** Monday. o **2.** Tuesday. o **3.** Wednesday. o **4.** Thursday. o **5.** Friday. o **6.** Saturday. 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.

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 number	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

Name	Value	Description
		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 *number* 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 drill-down, then you must configure each of the following applet user properties.

Name	Value	Description
Pane 1 Date Drilldown	source:destination	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*.
Pane 1 Item Drilldown Name	Time Scale Drilldown	Specify the drill-down object that resides in the applet that Siebel Open UI uses to display the third axis. You must also configure this drill-down 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

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*.

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*.
   - 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 in the topic *Customizing a Resource Scheduler*.
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.
Pane 2 Field *number*	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: `field name,abbreviated name` You can specify more than one field. For example, Pane 2 Field 1, Pane 2 Field 2, and so on.

Name	Value	Description
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 *Configuring Siebel Business Applications* .
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.
Pane 2 Bypass Overlap For Status	Dependency	Specify the type of events that Siebel Open UI does not split when an event overlap occurs. An *event overlap* 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. In this example, Siebel Open UI does not split any Dependency events that overlap.

Name	Value	Description
		You can use a comma to bypass multiple event types. For example, you can use the following value to bypass Dependency and Optioned events: Dependency,Optioned
Pane 2 Overlap Event LOV Type	TNT_SHM_INV_ STATUS	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.
Pane 2 Overlap Event Logical Order Values	Reserved, Option Reserved, Overbooked, Optioned, Unreserved, Unavailable, Unavailable Instance, Out of Order, Temporary	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: o Reserved o Option Reserved o Overbooked o Optioned o Unreserved o Unavailable o Unavailable Instance o Out of Order o Temporary
Pane 2 Round Minutes Events	15	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 extends 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.

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 a color for a Reserved event. Add each of the following applet user properties, as required.

Name	Value	Description
Pane 2 Color number	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.
Pane 2 Status LIC Field number	INVENTORY _STATUS,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.
Pane 2 Status LOV Type	TNT_FSD _COLOR_SCHEMA	Specify the color scheme that Siebel Open UI uses for events. To modify schemes, do the following: o Log in to a Siebel client with administrative privileges. o Navigate to the Administration - Data screen, and then the List Of Values view. o Query the Type Field for TNT_FSD_COLOR_SCHEMA. o Modify the fields, as necessary. 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 a color for the Prospect status in Siebel Hospitality. Configuration for your Siebel application might be different than it is for Siebel Hospitality.

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 number	DNMF,siebui-donotmove	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. Siebel Open UI uses this configuration for the icon. Use a comma to separate the abbreviation from the class name. You can configure more than one applet user property. For example, Pane 2 Icon 0, Pane 2 Icon 1, and so on.
Pane 2 Item Icon Fields	DNMF,NF,DF,SF, FSF,HF,AF,2HHF,SFF	Specify the abbreviations that you defined for the corresponding user properties in Step 3. 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. Use a comma to separate each abbreviation.

6. Configure Drag and Drop.

> **Note:** Drag and Drop functionality is a feature that you can enable or disable for the bar chart Schedule Pane in Siebel Open UI which either allows you to or prevents you from moving items around the Schedule Pane. You move an item by first selecting the item and (with the mouse button depressed) then moving the item elsewhere (and releasing the mouse button).

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	Specify to allow the user to select and move items. Use one of the following values: o **Y.** Allows you to select and move items. o **N.** Does not allow you to select and move items.
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	N/A	You can use this applet user property to specify another input argument that your deployment requires.
DragnDrop: Service Inputs 5	N/A	You can use this applet user property to specify another input argument that your deployment requires.
DragnDrop: Service Inputs 6	N/A	You can use this applet user property to specify another input argument that your deployment requires.
DragnDrop: Service Inputs 7	N/A	You can use this applet user property to specify another input argument that your deployment requires.

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", "CreateBookingRecord"	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.

Name	Value	Description
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: o **32.** Specifies the Month/Day-of-Week time scale. o **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*. 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", "CreateBookingRecord"	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.
Pane 2 Item Drilldown Name	Activity Drilldown	Specify the drill-down 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 drill-down object defines. This configuration works in conjunction with the DDID value that you configure in the Pane 2 Field *number* applet user property. You must configure the corresponding drilldown object in the applet.

Customizing Participant Availability in Resource Schedulers

This topic describes how to customize the controls that Siebel Open UI uses to display information about participant availability in a resource scheduler. You use custom cascading style sheet files to do some of this modification. For more information about how to organize these files, see *Organizing Files That You Customize*.

To customize participant availability in resource schedulers

1. Allow or disallow the user to resize the panes that Siebel Open UI uses to display information about participant availability:
   a. Log in to Siebel Tools.
   b. In the Object Explorer, click Applet.
   c. In the Applets list, query the Name property for Calendar GanttChart OUI Applet.
   d. In the Object Explorer, expand the Applet tree, and then click Applet User Prop.
   e. In the Applet User Props list, modify the following applet user property.

Name	Description
Disable Resize for Ganttchart	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. **Note:** This user property applies to all schedulers.

2. Modify the color that Siebel Open UI uses to display events:
   a. In the Object Explorer, click Business Service.
   b. In the Business Services list, query the Name property for Calendar Gantt Color Service.
   c. In the Object Explorer, expand the Business Service tree, and then click Business Service User Prop.
   d. In the Business Service User Props list, modify the following business service user property.

Name	Description
Event Status Mapping Color	For information about how to set this business service user property, see *Setting the Color for Events*.

3. Compile your modifications.
4. Modify the icons that Siebel Open UI uses to display information about participant availability. To do this, you can use one the following siebui-calgantt-icon CSS classes in your custom CSS file.

Description	Example

To modify the icon that Siebel Open UI uses for employees, use the siebui-calgantt-icon-employee CSS class.	`.siebui-calgantt-icon-employee {`
	`width: 16px;`
	`height: 16px;`
	`float: left;`
	`margin-top: 2px;`
	`background: url(../images/employees_icon.gif) no-repeat center center;`
	`}`

Description	Example
To modify the icon that Siebel Open UI uses for contacts, use the siebui-calgantt-icon-contact CSS class.	`.siebui-calgantt-icon-contactcall { width: 16px; height: 16px; float: left; margin-top: 2px; background: url(../images/contact_call.jpg) no-repeat center center; }`
To modify the icon that Siebel Open UI uses for resources, use the calgantt-icon-resource CSS class.	`.siebui-calgantt-icon-resource { width: 16px; height: 16px; float: left; margin-top: 2px; background: url(../images/resoure-items.gif) no-repeat center center; }`

5. Modify how Siebel Open UI displays information about the current record.

   You can use the .siebui-currentRecord CSS class in one of your custom CSS files. For example:

   ```
   .siebui-currentRecord {
    border-left: 3px solid green;
    border-right: 3px solid red;
    z-index: 1000;
   }
   ```

   This example modifies the class only for the current event. To change the default color for all events, modify the user property to the following:

   ```
   Pane 2 Event Default Color
   ```

6. Verify your work:

   a. Log into the client.
   b. On the Home page, click My Calendar.
   c. On the application-level menu, click Edit, and then click New Record.

      Siebel Open UI displays the eCalendar Detail View that contains the scheduling control.
   d. Verify that the resource scheduler includes the modifications that you configured in Step 2 through Step 5.

Setting the Color for Events

You can use the Event Status Mapping Color business service user property to set the color for each event type. It uses the following syntax:

```
"status_abbreviation,event_type:color_value"
```

where:

- status_abbreviation is defined in the Pane 2 Status LIC Field applet user property. Siebel Open UI uses this applet user property to display the scheduling control. In this example, you set *status_abbreviation* to GS (Gantt

Status). You can use any abbreviation. It is recommended that you use a short abbreviation, such as GS, to reduce the amount of information that Siebel Open UI must communicate.

- *event_type* specifies the type of event. For example, it can specify one of the following values:
  - Accepted
  - Declined
  - Not Responded
- *color_value* specifies a hexadecimal value that identifies the color that the cascading style sheet uses to display an event. For example, a *color_value* of #FF0000 specifies to display an event as red.

  You can use the following syntax to specify multiple color values:

```
"status_abbreviation,event_type:color_value;status_abbreviation,event_type:color_v
alue;"
```

  where:

- ; (semi-colon) separates each color value.

  For example, the following code sets the color for each event type:

```
"GS,Accepted:#d3ffd7;Declined:#6600CC;Not Responded:#000000"
```

  where:

  - `Accepted:#d3ffd7` sets the RGB color for Accepted events to light green (red at 82.75%, green at 100%, and blue at 84.31%).
  - `Declined:#6600cc` sets the RGB color for Declined events to purple (red at 40%, green at 0%, and blue at 80%).
  - `Not Responded:#000000` sets the RGB color for Not Responded events to black (red at 0%, green at 0%, and blue at 0%).

> **Note:** If you are setting the color for events in a Participant Availability scheduling control, the Business Service that requires modification is the Calendar Gantt Color Service. The value can be found in the `Pane 2 Event Color Service Name` user property in the applet.

For more information about how to use a hexadecimal number to represent a color, see the page about color codes at the ColorCodeHex website at http://www.colorcodehex.com.

Using CSS Classes to Set the Color for Events

You can use the following code instead of modifying the Calendar Gantt Color Service business service to set event colors:

```
siebui-calgantt-event_type
```

For example, you can add the following class to one of your custom CSS files to set the border color for Not Responded events to yellow:

```
.siebui-calgantt-NotResponded {
border: 1px solid #FFFF00;
}
```

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 in the topic *Customizing a Resource Scheduler*.
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 *number*	Quote Name Tip	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: **Event1** **Holiday resorts** **10:00** **12:00**
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 *number*	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.

Name	Value	Description
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 *number*	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: ○ Pane 2 Tooltip BC Name ○ Pane 2 Tooltip BO Name ○ Pane 2 Tooltip Field *number*
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: ○ Pane 2 Tooltip BC Name

Name	Value	Description
		○ Pane 2 Tooltip BO Name ○ Pane 2 Tooltip Field *number*
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: ○ Pane 1 Tooltip BC Name ○ Pane 1 Tooltip BO Name ○ Pane 1 Tooltip Field *number*
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: ○ Pane 1 Tooltip BC Name ○ Pane 1 Tooltip BO Name ○ Pane 1 Tooltip Field *number*
Pane 0 Tooltip Service Method	GetResTooltipInfo	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: ○ Pane 0 Tooltip BC Name ○ Pane 0 Tooltip BO Name ○ Pane 0 Tooltip Field *number*
Pane 0 Tooltip Service Name	TNT Gantt UI Service	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: ○ Pane 0 Tooltip BC Name ○ Pane 0 Tooltip BO Name ○ Pane 0 Tooltip Field *number*

7 Configuring Siebel UI to Interact with Other Applications

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*
- *Displaying Data from Siebel Open UI in External Applications*
- *Web Engine HTTP TXN Business Service*

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:

- *Siebel Portal Framework*
- *Integrating External Content*
- *Displaying Data from External Applications in Siebel Views*
- *Displaying Data from External Applications in Siebel Applets*

Siebel Portal Framework

This topic provides an overview of Oracle's Siebel Portal Framework and summarizes the technologies that make up the Portal Framework. It contains the following information:

- *Portal Framework Overview*
- *Portal Framework Architecture*

Portal Framework Overview

Enterprises are often composed of many different information technology resources, such as:

- Shared network directories.
- Department intranet sites.
- Legacy applications.
- Applications developed in-house.

- Purchased Web applications.

With many disparate applications and technologies, IT resources are difficult to maintain and difficult to use. For example, applications:

- Follow different user interface guidelines.
- Are rendered with different themes.
- Track profile attributes differently.
- Vary in the quality of online assistance.
- Have separate login and password credentials.
- Have different search functionality.

One solution to this problem is to integrate the various applications and content sources used in an enterprise and present them in a single user interface, called a portal. The Siebel Portal Framework allows you to do this. The Portal Framework provides you with the tools and supporting technologies that allow you to:

- Aggregate external data with Siebel data and present it in the Siebel user interface.
- Deliver Siebel CRM data to external applications.
- Integrate external application business logic and data with Siebel Business Applications.

Portal Framework Architecture

The portal framework includes the following framework components:

- Enterprise Application Integration
- Portal Agents that integrate external content into the Siebel user interface

Enterprise Application Integration

Siebel EAI provides mechanisms for sharing data and business logic with other applications, including:

- Integration objects
- Virtual business objects
- Programming APIs
- Predefined adapters and connectors

For more information about Siebel EAI, see *Overview: Siebel Enterprise Application Integration* and other EAI titles on Oracle Help Center.

Portal Agents

Portal Agents provide you with a mechanism to retrieve content from a non-Siebel source and display it in the Siebel user interface. The Portal Agent retrieves content on behalf of the user, logging on to the external application using the user's credentials and retrieving only the content that is targeted for the user. Portal Agents provide single sign-on capability and a profile tracking mechanism. For more information about Portal Agents, see *About Portal Agents*.

Integrating External Content

This topic provides an overview of Portal Agents. It describes the configuration and administration tasks necessary to display external content in the Siebel user interface. It also includes a reference topic that lists all of the commands available for use with Portal Agents. This chapter contains the following information:

- *About Portal Agents*
- *Process of Creating Portal Agents*
- *Determining the Login Requirements*
- *Portal Agent Configuration*
- *Portal Agent Administration*
- *Defining End-User Login Credentials*
- *Example Portal Agent*
- *Reviewing the SWE Log File*
- *Portal Agent Command Reference*

About Portal Agents

Portal Agents allow you to integrate external data into the Siebel user interface. Portal Agents retrieve data by sending HTTP requests to external applications, and then display the HTML results in a Siebel applet or on some other portion of a Siebel application Web page.

Portal Agents combine a set of features and technologies that allow you to integrate external content at the user interface layer, including the following:

- **Single sign-on technology (SSO).** For applications that are participating in a single sign-on framework, this feature eliminates the need for the user to enter login credentials, such as user name and password, more than once for each work session. For more information about single sign on, see *Siebel Security Guide* .
- **Session management and session reuse.** Allows the Siebel application and the external application to maintain a user's session context, without reauthenticating for subsequent requests. This minimizes session resource overhead on the external application, and allows the user to retain session context, such as shopping cart contents.
- **Time-out handling.** The Siebel Server automatically reauthenticates when a request is submitted after the external application's timeout period has passed.
- **Symbolic URLs, with multiple disposition types.** Allows content to be displayed in different ways, such as in a new browser window, inline with the other content, in an `<iframe>` tag. For more information, see *About Disposition Types*.
- **Session proxy.** For content integrated using a disposition type of Inline, the Siebel Server manages the interactions with external applications on behalf of the user. For more information about the Inline disposition type, see *Inline Disposition Type*.
- **Symbolic URL commands.** Commands that direct the Portal Agent to assemble the URL for the external application in several ways. These include dynamically referencing the user's user name and password, retrieving stored user name and password values, retrieving data from the user's personalization profile, establishing the size of an `<iframe>` tag, and determining whether to set the browser cookies from the application server's login page. For a complete list of commands, see *Portal Agent Command Reference*.

Note: Portal Agents do not integrate data at the data layer or integrate business logic. Other mechanisms in the Siebel Portal Framework, such as Integration Objects and Virtual Business Components, are designed to meet those types of integration needs. For more information about Siebel EAI, see *Overview: Siebel Enterprise Application Integration* .

Portal Agents and Authentication Strategies

Portal Agents can be configured to support different authentication strategies:

- **Simple Portal Agents.** The external application does not require any authentication parameters.
- **Single Sign-On Portal Agents.** The external application requires authentication parameters. Form-based Portal Agents send authentication parameters as part of the body portion of the HTTP request.

For more information about authentication, see *Siebel Security Guide* .

About Disposition Types

One of the steps in setting up a Portal Agent is creating a symbolic URL. The symbolic URL specifies the information necessary to construct the HTTP request to send to the external application. Symbolic URLs can be one of several disposition types. The disposition type determines the following:

- The interaction between the browser, the Siebel Server, and the external application.
- How external content is displayed in the user interface.

It is important to understand these disposition types and determine which one suits your integration needs. Each disposition type is discussed in one of the following sections.

For information about defining symbolic URLs, see *Defining Symbolic URLs*.

Inline Disposition Type

With a symbolic URL disposition type of Inline, the Siebel Server receives content sent by an external application. It combines the external content with Siebel-produced content and composes a single HTML page, which it then sends to the client browser for display to the user. Optionally, links in the aggregated content are rewritten so they reference the Siebel Server (proxy), rather than referencing the external application server directly. This allows the Siebel Server to handle links in the aggregated content in such a way that it appears to the user as one integrated application rather than from different application servers.

The Inline disposition type supports session management. The Siebel Server uses session management to manage session cookies and automatically log in again to an external application after a time out occurs.

The Inline disposition type requires that:

- The page you are integrating does not include complex JavaScript and does not reference frames.
- The maximum number of characters in the calling URL is 2048.
- No methods other than the `GET` method are invoked.

If the Inline disposition type is not appropriate, then you might try the IFrame disposition type.

IFrame Disposition Type

Use this disposition type when aspects of the external application do not allow content to be aggregated with other Siebel content. For more information, see *Portal Agent Restrictions*.

The IFrame disposition type uses the `<iframe>` tag to create an internal frame as part of the page generated by the Siebel Server. It allows the Portal Agent to retrieve content to populate the internal frame. This content does not pass

through the Siebel Server, but is directly requested by the client and sent by the application server to the user's browser. Although this disposition type is not as preferable as the Inline disposition type, in most cases, it is a method that works.

The IFrame disposition type supports JavaScript and frames. Therefore, if the Inline disposition type does not work, then the IFrame option is the best option. The IFrame disposition type also supports the Session Keep Alive feature. However, it does not support session management.

The IFrame disposition type works in many cases. However, it does not work when frames displayed within the `<iframe>` tag refer to top-level JavaScript objects. If frames in the page that you are trying to integrate refer to top-level JavaScript objects, then you might use the Web Control disposition type instead, if it is applicable.

Contextual Navigation Between Siebel CRM and Oracle Business Intelligence Pages

When an Oracle® Business Intelligence (Oracle BI) page is integrated with a Siebel application through the portal framework and the portal content is dependent on the Siebel record, any change or update of the record in the Siebel application must also be reflected in the portal content. For example, for an Oracle BI applet embedded in a view with the Account List applet, its content dynamically changes at the same time that the content is changed within the Account List applet. To enable this behavior, you must do the following:

- **Define a symbolic URL.** For more information, see *Defining Symbolic URL Arguments*.
- **Set parameters for the symbolic URL.** For more information, see *Portal Agent Command Reference*.

Form Redirect Disposition Type

In the Form Redirect scenario, the Siebel Web client submits a request to the Siebel Server. The Siebel Server creates a form with the necessary authentication information in it, and then sends the form back to the browser. The browser loads the form and then submits it to the external host for processing. The external host sends back the results, which the browser displays in a new window.

The Form Redirect disposition type is usually displayed in a new window, rather than inline with other Siebel applets.

The Form Redirect disposition type is not commonly used with Siebel CRM.

Portal Agent Restrictions

Portal Agents are meant to bring existing applications and content into the Siebel user interface without requiring additional modifications of the external application. However, this is not always possible due to the way HTML and Web browsers are designed. For example:

- The use of frames by an external application might not be amenable to inline aggregation methods.
- Specific frame references in the returned content referring to global frames (_NEW, _TOP, .parent()) might not be amenable to inline aggregation methods.
- Reliance on JavaScript functions defined in (assumed) external frames might not be amenable to inline aggregation methods.
- URLs that are created dynamically by JavaScript might not be amenable to any fixup techniques, because the URLs would not be easily parsed on the HTML content.

For these reasons, an Inline disposition type does not work often. However, if you control both the Siebel application instance and the external application, and can resolve some of these issues, then the Inline disposition type might work correctly. For more information about the Inline disposition type, see *Inline Disposition Type*.

If you do not have control over the external application, the IFrame disposition type is the method most likely to provide satisfactory results. It works with about 80% of the form-based application sites tested. For more information about the IFrame disposition type, see *IFrame Disposition Type*.

Disposition Types Summary

The following table summarizes the characteristics of each disposition type.

Disposition Type	Benefits	Limitations
Inline	• Inline integration with the Siebel user interface. • Session management, including managing session cookies and automatic re-login after time out. • Opens an external URL in a new popup window.	• Only works in very few cases. • Does not work with complex JavaScript. • Does not work if there are reference to frames. • Supports the `GET` method only. • URL limited to 2048 characters.
IFrame	• Inline integration with the Siebel user interface. Supports complex JavaScript. • Supports references to frames. • Session Keep Alive supported. • Works for most cases.	• No session management. • Does not support frames that reference top-level JavaScript objects. • Does not open an external URL in a popup window.

Process of Creating Portal Agents

To create a Portal Agent, perform the following tasks:

1. *Determining the Login Requirements*.
2. *Configuring Business Components to Handle External Data*.
3. Complete one of the following in the following sections:
    - *Displaying External Content Within an Applet*
    - *Displaying External Content Outside of an Applet*
4. *Defining Web Applications*
5. *Defining Symbolic URLs*
6. *Defining Symbolic URL Arguments*

Determining the Login Requirements

Before you configure Portal Agents, you must understand what information is required by the external application to authenticate users. Typically, this information is gathered using a form page, also called a login page, and then sent to the external application. You must determine exactly what information the form gathers from the user and sends to the external application, including field names and values.

In cases where you have specific knowledge about how an external application is implemented and can consult with authoritative sources regarding how the application authenticates users, determining the required input fields and values is relatively simple.

In cases where you do not have specific knowledge about how an external application is implemented, you must attempt to understand its authentication method by examining the application's login page. The steps describe an approach that you can use to reverse-engineer a login page and provide related Portal Agent configuration tips.

Note: It is not always possible to reverse-engineer a login page. For example, JavaScript might process login field values prior to delivering the `POST` back to the application server, session values might be encoded in the form itself, or session values might be stored in the browser's session cookies.

This task is a step in *Process of Creating Portal Agents*.

To reverse-engineer a login page

1. Navigate to the external application's login page and determine whether the external application uses authentication.

   For more information, see *Defining Symbolic URLs*.
2. If the external application uses form-based authentication, then view the login page's HTML using your browser's view source command.
3. Identify the form on the login page that asks for user credentials (the form might ask for other information as well) and identify the input fields in this form used to authenticate users.

   It is usually best to strip out all non-form lines of HTML and to isolate the `<input>` tags. That is, remove lines previous to `<form ...>` and after `</form>` and remove lines that are not part of the `<input>` tags.
4. Determine whether the method attribute of the `<form>` tag is `POST`.

   If it is `POST`, then you must define the PostRequest command as an argument of the symbolic URL. For more information, see *Defining Symbolic URL Arguments* and *PostRequest Command*.

   If it is `GET`, then you do not have to define a symbolic URL command, because the default method of symbolic URLs is `GET`.
5. Determine the target of the form's action attribute, which is usually specified as `action=" some string "`.

   If the target of the action attribute is an absolute URL, one that begins with `http` or a forward slash (`/`), then use this URL as the base of the Portal Agent.

   If it is a relative address, then you also have to determine where the root of the URL is defined. It could be defined relative to the URL of the login page itself (most common), in a `<codebase>` tag (rare) or in JavaScript (hard to determine).

   The target URL is defined using the Host Administration View and the Symbolic URL Administration view. For more information, see *Defining the External Host* and *PostRequest Command*.
6. Determine any argument values defined in the target URL.

   These are the characters after the *?* character. Usually, these are simple field-value constants. The exception is when a field or a value is a session identifier that is dynamically assigned by the external application server and is only valid for a period before it times out. In this case, it might not be possible to configure a Portal Agent. Define any argument values contained in the target URL as symbolic URL arguments. For more information, see *Defining Symbolic URL Arguments*.

7. Identify each of the form's `<input>` tags and determine which ones are necessary to send to the external application for authentication.

   Often there are `<input>` tags in the form with a `type` attribute of `hidden` that are not evident when interacting with the application. Determining whether hidden fields are optional or required is often process of trial and error.

   Some `<input>` tags might not have values identified. Either these fields are awaiting input to be entered by the user (for example, login name or password) or they are hidden fields with no values.

   - If the input field is specific to the user (it asks for the user's login name and password), then you can use *UserLoginId Command* and *UserLoginPassword Command* commands to instruct the Portal Agent to retrieve the user's credentials from the user's My Logins view. For more information, see *Defining End-User Login Credentials*.
   - If there are hidden fields with no values, then, when you enter them as symbolic URL arguments, make sure that the Required Argument column is not checked. If it is checked, and the input field has no value, then the Portal Agent does not send this request to the target application server, because there is no value to put in its place.

   You define the input fields and values as symbolic URL arguments. For more information, see *Defining Symbolic URL Arguments*.

Note: The Mozilla browser includes a page info command (^I) that analyzes forms on a page and displays the method, input fields, and so on.

Portal Agent Configuration

Using Portal Agents to integrate external content into the Siebel user interface requires some simple configuration in Siebel Tools. You must configure a field on the business component to handle external data and then configure either an applet or a Web page item to display the content in the user interface. An applet displays external content inside the applet container on a view. A Web page item displays external content outside of an applet, such as in the banner frame for example.

Note: This topic describes the configuration tasks that are unique to integrating external content with the Siebel user interface. It does not describe standard configuration tasks that you might be required to perform. For example, after you configure an applet to display external content, you might have to associate that applet with a view, add the view to a responsibility, and so on. These additional tasks are standard procedures for configuring Siebel Business Applications and are outside the scope of this book. For more information about configuring Siebel Business Applications, see *Configuring Siebel Business Applications* .

Configuring Business Components to Handle External Data

To configure business components to handle external data using a symbolic URL, you must create a new calculated field on the business component. Rather than representing structured content, such as records in a database, this field represents the HTML content sent from an external host.

Note: Although a symbolic URL displays data that is not stored in the database, the business component must have at least one record stored in an underlying table so that it is instantiated at run time.

This task is a step in *Process of Creating Portal Agents*.

To configure a business component to handle external data using a symbolic URL

1. Create a new field on the business component.
2. Set the field's Calculated property to TRUE.
3. Set the field's Type property to DTYPE_TEXT.
4. In the Calculated Value field, enter the name of the symbolic URL (enclosed in double quotes) that you want to use to submit the HTTP request.

   The name of the symbolic URL in the Calculated Value field must be enclosed in double quotes so that it evaluates as a constant. See the business component named *AnalyticsSSO* in the Siebel Repository for an example of fields configured this way.

Displaying External Content Within an Applet

After you have created the calculated field on the business component, you expose it in the user interface. You display the external content using a control in a form applet or list applet.

> **Note:** You can also expose external content outside an applet, such as in the banner area. See *Displaying External Content Outside of an Applet*.

This task is a step in *Process of Creating Portal Agents*.

To display external content within an applet

1. Create an applet that you want to use to display the external content.

   The applet must be based on the business component that you configured in *Configuring Business Components to Handle External Data*.
2. Add a new control or list column to the applet.
3. Associate the control or list column with a calculated field on the business component that is configured to represent the external data.
4. Set the control or list column's Field Retrieval Type property to Symbolic URL.
5. Set the control or list column's HTML Type property to Field.

Displaying External Content Outside of an Applet

After you have created the calculated field on the business component, you expose it in the user interface. You can display the external content outside of an applet using Web Page Items.

> **Note:** You can also expose external content inside an applet, by using an Applet Control or List Column. For more information, see *Displaying External Content Within an Applet*.

This task is a step in *Process of Creating Portal Agents*.

To display content outside of an applet

1. Start Siebel Tools.
2. Go to the Web Page object type and select the Web page on which to display external data.
3. Create a new Web Page Item or use an existing one.
4. Set the Type property of the Web Page Item to Field.

5. Create the following two Web Page Item Parameters:

Name	Value
FieldRetrievalType	Symbolic URL
SymbolicURL	[name of symbolic URL]

> **Note:** The symbolic URL is mapped to the calculated field defined for the business component.

Portal Agent Administration

You administer Portal Agents through several views located under the Administration - Integration screen in the Siebel Web client. As described in the following topics, these views allow you to define how to handle links, define the external host, and define the HTTP request that is sent to the external host.

Defining the External Host

You define the external data hosts in the Host Administration view. This view allows you to do the following:

- Maintain external host names in a single place.
- Define how to handle (fix) links after external HTML content is rendered.

To define a data host

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Host Administration.
3. Enter a new record and define the necessary fields.

   Some of the fields are described in the following table:

Field	Comments
Name	Name of the external host.
Virtual Name	User-defined name for the host.
Authentication Type	Leave this value blank. For more information, see *Defining Symbolic URLs*.

Defining Web Applications

Web applications allow multiple symbolic URLs to send requests to the same Web application and share the same session. This is useful if you have two different applet controls that use symbolic URLs to submit requests to the same Web application. You can associate these symbolic URLs to a single Web application and specify whether they share the same session.

There might be cases in which you do not want requests to share the same session. For example, you might not want to share a session when a session cookie contains more information than the session ID, as this could result in unexpected behavior. When you define a Web application, you specify whether it shares sessions.

Web applications also allow you to define the Time Out value for the session time out feature. The Session Time Out feature is only applicable to symbolic URLs with a disposition type of Inline.

This task is a step in *Process of Creating Portal Agents*.

To define a Web application

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Web Application Administration.
3. Enter a record and complete the fields.

   Some of the fields are described in the following table:

Field	Description
Shared	Indicates whether requests generated by symbolic URLs associated with this Web application share the same session.
Time Out	Defines the time out parameter for the session management feature, which is only applicable to symbolic URLs with a disposition type of Inline.

Defining Symbolic URLs

You use the Symbolic URL Administration view to specify how to construct the HTTP request to the external application and to define any arguments and values to be sent as part of the request.

This task is a step in *Process of Creating Portal Agents*.

To define a symbolic URL

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Symbolic URL Administration.
3. In the Symbolic URL Administration list view, enter a new record.

   Some of the fields are defined in the following table:

Field	Description
URL	Use the URL field to enter a URL for the external application. A best practice is to substitute the host's Virtual Name, the one that you defined in the Host Administration view, for the host's actual name. Doing this makes administering host names easier, because you might have many symbolic URLs pointing to one host. If the host name changes, then you only need to change it in the Host Administration applet rather than having to change it in several symbolic URL definitions. For example:

Field	Description
	`https://Virtual_Host/path...` When using a Symbolic URL with the *Inline* disposition type and HTTPS as the protocol, you must make sure that the certificates offered by the external application are: o Not expired o Trusted by the Siebel application, refer *Configuring Siebel Server and Config Agent for SHA2 Outbound* o Have the same domain as the URL which you are calling If any of these conditions are false, you will receive an error, and the external data will not be rendered in the View. **Note:** If you do not use HTTPS in your URL, then the behavior is the same as it has always been. Oracle recommends using HTTPS in your URL. **Note:** Use the Secure Sockets Layer protocol (SSL) with symbolic URLs to ensure that communication is secure. For more information about using SSL, see Siebel Security Guide . For applications that use form-base authentication, the URL is identified by the action attribute of the `<form>` tag. For more information, see *Determining the Login Requirements*.
Host Name	The Virtual Name of the host defined in the Host Administration view.
Fixup Name	Name of the fixup type defined in the Fixup Administration view. The fixup type defines how links embedded in the external HTML content are rendered. For example: o **Default.** Use this fixup type with the IFrame disposition type. Link fixup is inside the view. This fixup does not affect any of the links. The links (relative or absolute) remain as they are with the content being passed back in its original form. o **InsideApplet.** This fixup converts all of the relative links to absolute links and any links using a host defined in the Host Administration view are proxied in order to maintain SWE context. o **OutsideApplication.** This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied.
Multivalue Treatment	Determines how arguments are handled. Possible values are: o **Comma Separated.** Instructs SWE to insert a comma between the values defined in the symbolic URL arguments when appending the arguments to the URL. It inserts a comma after the value in the first Argument Value field and the first value in the second Argument Value field. The second Argument Value field is simply a text string entered by the user. o **Separate Arguments.** Instructs SWE to enter separate arguments for each value defined in the two Argument Value fields. o **Use First Record Only.** Uses the first record in the current record set.
SSO Disposition	The value selected in this field determines how the HTTP request is constructed and sent and how the external content is rendered in the user interface. Possible values are:

Field	Description
	○ **Inline.** Proxies the request through the Siebel Server and displays content inline with other applets on a view. ○ **IFrame.** Uses the `<iframe>` tag to display content inline with other applets on a view. ○ **Web Control.** Browsers displaying symbolic URLs of type Web Control must be set to handle ActiveX controls. For more information about browser security settings, see *Siebel Security Guide* . ○ **Form Redirect.** SWE constructs a form which it sends back to the browser, which the browser then sends to the external host. The content received is displayed in a new window. ○ **Server Redirect.** SWE sends the browser a 302 Response with the value of the external host's URL in the header. The browser is redirected to the external host. The content received is displayed in a new window. Note that for Server Redirect there is a required Symbolic URL argument. For more information, see *Portal Agent Restrictions*. For detailed descriptions of each disposition type, see *About Portal Agents*.
Web Application Name	Associates a Web application with this symbolic URL. For more information about Web applications, see *Defining Web Applications*.

Defining Symbolic URL Arguments

Symbolic URL Arguments allow you to configure Portal Agents in several ways. You use symbolic URL arguments for two purposes, to define data to be sent to an external host and to submit commands to SWE that affect the behavior of Portal Agents.

When defining arguments that send data, such as authentication requirements, the Argument Name and Argument Value are appended to the URL as an attribute-value pair. You can define symbolic URL arguments that send data as constants or that dynamically retrieve data from the Siebel database. Symbolic URLs allow you to retrieve data from the user's instantiated Siebel business component, such as Service Request or Account, or retrieve data from the Siebel Personalization business component, such as the user's ZIP Code or Language.

For information about how to determine required data for applications that use form-based authentication, see *Determining the Login Requirements*.

Symbolic URL arguments also allow you to implement commands, which you use to define the behavior of Portal Agents. For usage descriptions of available commands, see *Portal Agent Command Reference*.

This task is a step in *Process of Creating Portal Agents*.

To define symbolic URL arguments

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Symbolic URL Administration.
3. In the Symbolic URL Administration list view, select the symbolic URL that you want to configure.

4. In the Symbolic URL Arguments form, enter the arguments that need to be sent to the external host.

   Some of the fields are defined in the following table:

Field	Description
Name	Name of the argument. For arguments of type Constant, Field, and Personalization Attribute, this field defines the exact field name expected by the external application. It is the first part of an attribute-value pair appended to the URL. For argument types of commands, the Name can usually be anything. The only exception to this is for the EncodeURL and PreloadURL commands. For more information, see *Portal Agent Command Reference*.
Required Argument	When this field is checked (default) the argument must have a value. If you are configuring an argument that does not have a value, then uncheck the Required field. If an argument has no value and the Required field is checked, then the request is not sent because there is no value to append to the URL.
Argument Type	The Argument Type determines the source of the data to be send along in the HTTP request. Possible values are: o **Constant.** Sends the value defined in the Argument Value field in the request. o **Field.** Sends the value of a single-value or multi-value field from the current Siebel business component. o **Profile Attribute.** Sends the value of a field from the Siebel Personalization business component. o **URL Argument.** Data comes from the named argument of the current request. o **Language Value.** The user's current language setting; for example, ENU. o **Command.** Implements commands that allow you to affect the behavior of the symbolic URL. For a complete list of commands, see *Portal Agent Command Reference*.
Argument Value	The value of the argument varies depending on the Argument Type. Descriptions of possible values for each argument type are described here. If the Argument Type is one of the following: o **Constant.** The Argument Value is the second part of the attribute-value pair that is appended to the URL. o **Field.** The Argument Value defines a field name from the current business component. The data from this field is the second part of an attribute-value pair that is appended to the URL. o **Profile Attribute.** The Argument Value defines a field name on the Siebel Personalization business component. The data from this field is the second part of an attribute-value pair that is appended to the URL o **URL Argument.** The Argument Value defines the name of the argument on the incoming SWE request. o **Language Value.** The Argument Value is left null. o **Command.** The Argument Value typically defines the name of the command. For more information, see *Portal Agent Command Reference*.

Field	Description
Append as Argument	When this field is checked (default), the value is added as a URL argument on the outgoing request. If this field is not checked, then the value is substituted in the text of the outgoing URL.
Sequence	Determines the sequence of the arguments. In some cases the target host requires arguments in a particular order.

Configuring Multiple Symbolic URLs and Hosts for Alternative Execution Locations

You can configure multiple symbolic URLs and symbolic URL hosts, to execute applications in alternative locations (for example, for testing or demonstration purposes).

> **Note:** When you use an alternative symbolic URL host, all symbolic URLs in the application that are configured to use that host will use the alternative host name. In contrast, when you use alternative symbolic URLs, each symbolic URL used in the application must have its own alternative symbolic URL. Therefore, you can reduce the effort required to execute the application in an alternative location by using an alternative symbolic URL host rather than a symbolic URL.

Configuring Alternative Symbolic URLs

To use an alternative symbolic URL, define the additional symbolic URL at the Symbolic URL Administration view, and specify the following parameter in the [DataSources] section of the application's configuration file:

SymbolicURLSuffix. The value of this parameter is appended to the end of the name of the default symbolic URL to specify the name of the alternative symbolic URL.

For example, if the parameter SymbolicURLSuffix is set to _MyDemo in the application's configuration file, and the default symbolic URL name is AccountNews, then the symbolic URL that is used when the application is executed is AccountNews_MyDemo. The URL value associated with the AccountNews_MyDemo symbolic URL in the Symbolic URL Administration page is used.

> **Note:** When you define the alternative symbolic URL, its name must match the name of the existing symbolic URL with the value of the SymbolicURLSuffix parameter appended to it.

For more information about defining symbolic URLs, see *Defining Symbolic URLs*.

Configuring Alternative Symbolic URL Hosts

To use an alternative symbolic URL host, define the additional symbolic URL host at the Host Administration view, and specify the following parameter in the [DataSources] section of the application's configuration file:

SymbolicURLHostSuffix. This value is appended to the end of the name of the existing symbolic URL host to specify the name of the alternative symbolic URL host.

For example, if the parameter SymbolicURLHostSuffix is set to _demo in the application's configuration file, and the existing host name is ABC, then the new host name is ABC_demo. The host name value associated with ABC_demo in the Host Administration page is used.

Note: When you define the alternative symbolic URL host, its name must match the name of the existing symbolic URL host with the value of the SymbolicURLHostSuffix parameter appended to it.

For more information about defining hosts, see *Defining the External Host*.

Defining Content Fixup

The Fixup Administration view allows you to define how links embedded within external HTML content are rendered in the Siebel user interface. The fixup types that you define here will be associated with symbolic URLs.

To define a fixup type

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Fixup Administration.
3. Enter a new record and define the fields.

   Some of the fields are described in the following table:

Field	Comments
Link Context	Select one of the following values: ○ **Do Nothing.** This fixup does not affect any of the links. The links (relative or absolute) remain as they are with the content being passed back in its original form. ○ **Outside Application.** This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied. ○ **Inside Application.** This fixup converts all of the relative links to absolute links and any links using a host defined in the Host Administration view are proxied in order to maintain SWE context. After the user clicks a link, this fixup type renders HTML in the view, using the entire view for display. ○ **Inside Applet.** This fixup handles links the same way as the Inside Application fixup type. However, in this case, when a user clicks a link, it renders HTML within an applet. The other applets remain present on the view.
Context View Name	Name of the view that displays the link. This is optional.
Link Target	Specifies the name of a specific target frame of the link. For example, *_blank* for a new browser window or *AnyName* to open a window of that name. This option is not often used.

Note: Fixup is required for all links.

Defining End-User Login Credentials

The Portal Framework provides a mechanism to store user login credentials for external Web applications. The SSO Systems Administration view allows you to specify an external application and then enter login credentials on behalf of users. The My Logins view, located in the User Preferences screen, is used by end users to maintain their own credentials.

To specify an external Web application and define login credentials

1. Navigate to the Administration - Integration screen, and then SSO Systems Admin List.

2. In the SSO Systems list, enter a new record and define the following:

Field	Description
System Name	Name of the external Web application.
Symbolic URL Name	Select the name of the symbolic URL that interacts with the external Web application. The symbolic URL must be configured with the *UserLoginId Command* and *UserLoginPassword Command* commands as arguments. These arguments instruct the symbolic URL to pass the stored login credentials when authenticating with an external Web application.
Description	Enter a description of the Web application.

3. If you are defining login credentials on behalf of end users, then, in the SSO System Users list, enter end-user login names and passwords.

Example Portal Agent

This topic provides an example of using a symbolic URL to integrate content from an external site.

Each of these steps is described in the topics that follow. This example uses www.example.com, which does not have the login page and other elements described here; substitute your actual site.

Note: This example assumes that the underlying objects are already configured to support the symbolic URL. For more information, see *Portal Agent Configuration*.

Review the Login Form

By reviewing the login page at www.example.com, you can determine the target URL of the Action attribute and the required arguments that are being passed to the Web application. Assume that www.example.com has a login page that contains the following `<form>` and `<input>` tags:

```
<form action="/index.shtm" method="POST" name="frmPassLogin" onsubmit="return
logincheck();">
 <input TYPE="TEXT" NAME="SearchString" SIZE="18" MAXLENGTH="100" VALUE="">
 <input type="hidden" value="All" name="sc">
 <input type="hidden" value="ON" name="FreeText">
 <input type="image" src="/images/nav/button/bttn_form_arrow.gif" NAME="Action"
 border="0"/ alt="Submit Search"></td>
 <input type="text" name="username" size="18">
 <input type="password" name="password" size="18">
 <input type="image" src="/images/nav/button/bttn_form_arrow.gif" border="0"
 name='login' />
 <input type="checkbox" name="remember" checked/> <span
 class="bdDkGray">Remember my Login<br></span>
</form>
```

From the `action` attribute of the `<form>` tag, you can determine that the target URL is relative to the root of the login page's URL. Therefore, the target URL is:

```
www.example.com/index.shtm
```

You can also determine that the `method` attribute of the `<form>` tag is `POST`:

```
method="POST"
```

After reviewing the `<input>` tags, you can determine that the required arguments are:

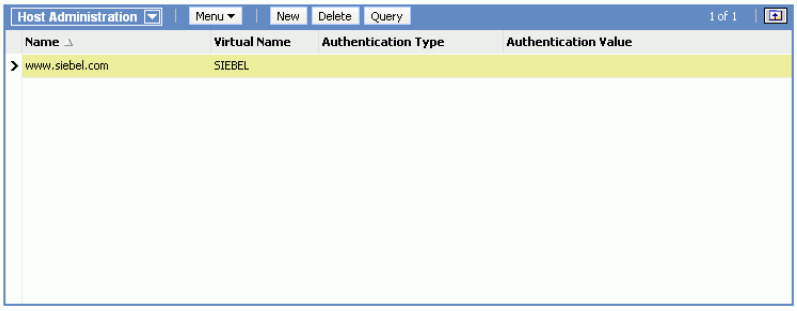
```
username
password
```

Note: Notice that not all input fields are necessary for login. For more information about reviewing login forms, see *Determining the Login Requirements*.

Define the External Host

The external host is simply the address of the login page. In this example, it is www.example.com. Be sure to provide a meaningful name in the Virtual Host Name field. This value is used instead of the actual host name when you define the symbolic URL. This makes administration easier if the host name changes. Also notice that there is no value for the Authentication Type.

The following figure shows the external host defined for this example where Name is www.siebel.com and Virtual Name is SIEBEL.

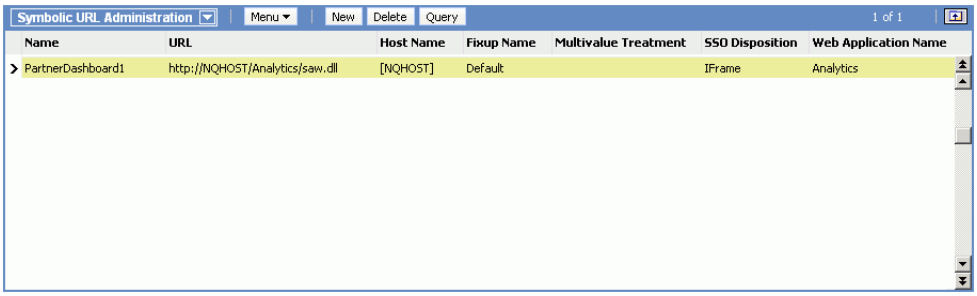


For more information, see *Defining the External Host*.

Define the Symbolic URL

After you define the external host, you can define the symbolic URL. Notice that the URL defined here uses the Virtual Name of the host, not the actual name. Also notice that, when you select the external host from the Host Name field, it is populated with the actual host name. When SWE constructs the URL, it substitutes the actual Host Name for the Virtual Name in the URL. In this example, the fixup type is Default, because the page is displayed in the browser using the `<iframe>` tag and therefore, it is recommended that links not be fixed up in any way.

The following figure shows the symbolic URL defined for this example where Name is PartnerDashboard1, URL is http://NQHOST/Analytics/saw.dll, Host Name is [NQHOST], Fixup Name is Default, SSO Disposition is IFrame, and Web Application Name is Analytics.



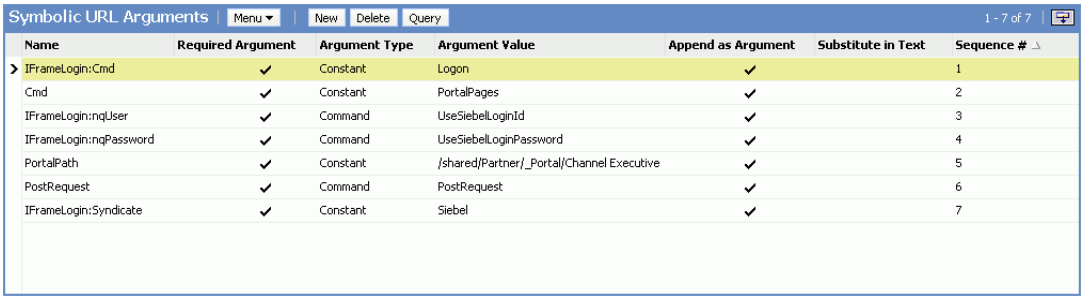
For more information about defining symbolic URLs, see *Defining Symbolic URLs*.

Define Symbolic URL Arguments

You use symbolic URL Arguments to define the information that you want to append as arguments to the URL. You also use symbolic URL arguments to define commands that you want to execute. In this case, the following arguments are required:

- **PostRequest.** This command instructs SWE to submit the request using a `POST` method rather than `GET`, which is the default. In this case, you know that `POST` is required because the method attribute of the `<form>` tag specifies `POST`.
- **UserLoginPassword.** This command instructs SWE to retrieve the password stored for the user and pass it to the external application. The name of this argument is the name of the input field expected by the external application. In this case, it is *password*.
- **UserLoginID.** This command instructs SWE to retrieve the stored login name for the user and pass it to the external application. The name of this argument is the name of the input field expected by the external application. In this case, it is *username*.

The following figure shows the symbolic URL arguments defined for this example, which are as follows: IFrameLogin:Cmd, Cmd, IFrameLogin:nqUser, IFrameLogin:nqPassword, PortalPath, PostRequest,a dn IFrameLogin:Syndicate.



For more information about symbolic URL arguments, see *Defining Symbolic URL Arguments*. For more information about symbolic URL commands, see *Portal Agent Command Reference*.

Define User Login Credentials

Finally you must define login credentials for a user. The values defined here are appended as arguments to the URL constructed by SWE. In this case, the following user name and password are defined:

- The user name is Joe_Smith@example.com.
- The password is abracadabra.

Testing the Integration

After completing the previous steps, you can test the integration.

To test the integration

1. Log out of the application.
2. Log back in as the test user.
3. Navigate to the applet or Web page item that is associated with the symbolic URL.

   Content from the external host, in this case example.com, is displayed in the Siebel user interface.

Reviewing the SWE Log File

Reviewing the SWE log file can help you to debug errors in your Portal Agent configuration.

- The location of the log file is `SIEBSRVR_ROOT\log`.
- The name of the log files are swelog_pid.txt and sweusage_pid.txt, where pid is the process ID of the corresponding Siebel process.

For more information about log files and about configuring log levels, see *Siebel System Monitoring and Diagnostics Guide* .

Portal Agent Command Reference

Portal Agent commands allow you to carry out actions such as: use a set of stored credentials for authentication or define additional attributes for the `<iframe>` tag. These commands are entered as symbolic URL arguments. For more information, see *Defining Symbolic URLs*.

The commands are described in the following subtopics.

EncodeURL Command

Use the EncodeURL command to specify whether to encode arguments appended to the symbolic URL. By default, the URL is encoded. However, some servers do not recognize standard encoding, in which case you can use this command to not encode the URL.

Define the fields in the Symbolic Arguments applet. See the following table.

Field	Value
Name	EncodeURL
Argument Value	*TRUE* or *FALSE*

FreePopup Command

Use the FreePopup command to show portal contents in a popup window.

The symbolic URL contains the FreePopup command, it notifies the client that the pop-up is a free one and the client displays the contents in the pop-up window.

FreePopup is supported for FormRedirect, the only disposition type available for opening a portlet in a pop-up.

To start the external application as a full browser window, use the values in the following table.

Name	Required Argument	Argument Type	Argument Value	Sequence	Append as Argument
FreePopup	True	Command	True	1	True
FullWindow	True	Command	True	2	True

To start the external application as a modal window, use the values in the following table.

Name	Required Argument	Argument Type	Argument Value	Sequence	Append as Argument
PopupSize	True	Command	750x500	1	True
FreePopup	True	Command	True	2	True

To start the external application in a new browser tab (for each click), use the values in the following table.

Name	Required Argument	Argument Type	Argument Value	Append as Argument
FreePopup	Y	Command	True	Y
Target	Y	Command	_blank	Y

IFrame Command

Use the IFrame command to define additional HTML attributes for the `<iframe>` tag.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value	Example
Name	Any Name	None
Argument Value	IFrame [*attribute*]=[*value*]	IFrame Height=100 Width=500

Field	Value	Example

Disposition Types

Use the IFrame command with the IFrame disposition type.

IsRecordSensitive Command

Use the IsRecordSensitive command to turn on or off the record-sensitive feature. Set the value to TRUE to ensure that a child applet with a symbolic URL is refreshed on the parent record, for instance, when you embed an Analytics report as a child applet with a requirement that it display contextual information.

This command is turned off by default. Set this argument value to TRUE in the Symbolic URL Arguments configuration.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	IsRecordSensitive
Argument Value	TRUE

NoCache Command

Use the NoCache command to instruct SWE not to cache Inline responses on the server. This command is only valid for the Inline disposition type.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	Any name
Argument Value	NoCache

NoFormFixup Command

Use the NoFormFixup command to instruct SWE not to fix up a form by putting proxy SWE arguments into links that appear on the page.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	Any name
Argument Value	NoFormFixup

Field	Value

PreLoadURL Command

Use this command to specify a preloaded URL. Use this command when the external application gathers information from a preloaded cookie on the client machine. Use this command with disposition types of IFrame and Web Control.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	PreLoadURL
Argument Value	[*URL*]

PostRequest Command

Use PostRequest to configure the Portal Agent to use the POST method instead of the GET method, which is the default. Use this command when the method of the action attribute is POST. This method avoids displaying user information on a Web page or browser status bar. Use this command with disposition types of IFrame and Web Control only.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	Any Name
Argument Value	PostRequest

UserLoginId Command

Use the UserLoginId command to send the stored user login ID for a particular Web application. The command gets the user's Login ID from the My Login Credential business component.

For more information about how user login IDs are entered into this business component, see *Defining End-User Login Credentials*.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	[*input field name*]
Argument Value	UserLoginId

UserLoginPassword Command

Use the UserLoginPassword command to send the stored user password for a particular Web application. The command gets the user's password from the My Login Credential business component.

For more information about how user passwords are entered into this business component, see *Defining End-User Login Credentials*.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	[*input field name*]
Argument Value	UserLoginPassword

UseSiebelLoginId Command

Use the UseSiebelLoginId command to retrieve the user's Siebel login ID from the stored set of credentials.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	[input field name]
Argument Value	UseSiebelLoginId

UseSiebelLoginPassword Command

Use the UseSiebelLoginPassword command to retrieve the user's Siebel password from the stored set of credentials.

Define the fields in the Symbolic URL Arguments applet. See the following table.

Field	Value
Name	[*input field name*]
Argument Value	UseSiebelLoginPassword

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 b to this new opportunity.
   h. Click New to create another new opportunity, and then add the contact that you created in Step d 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. Download the sociallyawarepmodel.js file into the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\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. For more information about the language_code, see *Languages That Siebel Open UI Supports*.
3. Use a JavaScript editor to open the sociallyawarepmodel.js file that you downloaded in Step 2.
4. Locate the following code:

   ```
   SociallyAwarePM.prototype.Init = function(){
    SiebelAppFacade.SociallyAwarePM.superclass.Init.call(this);
   ```

5. Add the following code immediately under the code you located in Step 4:

   ```
   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.
6. Add the following code immediately after the code you added in Step 5:

   ```
   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 7.

7. Add the following code immediately after the FetchConnectionFromLinkein 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* and *GetActiveView Method*.

8. Add the following code immediately after the AddProperty methods you added in Step 5:

```
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.

9. Configure the manifest:
   a. Log in to a Siebel client with administrative privileges.

      For more information about the screens that you use in this step, see *Configuring Manifests*.
   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
```

10. 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.

> **Note:** When a symbolical URL is displayed in the Siebel Web framework, Siebel Open UI sends it regular ping requests to prevent the session from timing out. This is done because Siebel Open UI can not detect activity, or lack of activity, on the symbolic URL

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: `SymbolicURLGoogleMap`
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: `SymbolicURLGoogleMap` 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 configuration, you create an applet that Siebel Open UI can to 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 2b, 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

Property	Value
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 move 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`

Field	Value
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.
Append as Argument	Y
Substitute in Text	N
Sequence#	1

f. In the Symbolic URL Arguments list, add the following symbolic URL argument. Siebel Open UI uses this argument to embed the Google map in the applet.

Field	Value
Name	output

Field	Value
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*
- *Configuring Multiple Siebel Portlets in Portal Applications*
- *Configuring Advanced Options*
- *Configuring Communications with Siebel Portlets When Hosted Inside iFrame*
- *Additional Considerations*
- *Limitations*
- *Preparing Standalone Applets*
- *Using iFrame Gadgets to Display Siebel CRM Applets in External Applications*
- *SWE API*

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

You can configure Siebel Open UI to display a Siebel portlet. A *Siebel portlet* is a Siebel Open UI application that is embedded in a thirty-party website. Oracle WebCenter and iGoogle are examples of these types of third-party websites. An HMTL iFrame is used in these websites to display part of the Siebel application in a portlet window.

This topic describes how to display Siebel portlets in external applications.

Configuring Siebel Open UI to Consume Siebel Portlets

Siebel portlets can be integrated inside a portal application using iFrame or any other mechanism supported by the portal application. Siebel accepts both GET and POST requests.

To make a Siebel Server available as part of a portal, you can add the server URL to an iFrame that resides on the main Web page. In this sample code, the HTTP GET method is used:

```
<HTML>
 <BODY>
 <IFRAME src = "http://server_address/siebel/app/application/
lang?SWECmd=SWECmd=GotoView&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/siebel/app/callcenter/enu?
SWECmd=GetApplet&SWEApplet=Quote+List+Applet&IsPortlet
=1&SWESM=Edit+List"style="height: 50%;width: 100%;&KeepAlive=1&PtId=my_theme"
```

The following table describes the parts of this iFRAME src that specifies the Siebel URL.

URL Argument	Description
http://*server_name*.example.com	Access the Siebel Server that resides at *server_name*.example.com.
/callcenter/enu	Run the CallCenter application.
SWECmd=GetApplet	Provide commands to the Siebel Web Engine.

URL Argument	Description
SWEApplet=Quote+List+Applet	Display the Quote List Applet.
IsPortlet =1	Run the CallCenter application as a portlet.
SWESM=Edit+List	Use the Edit List Mode
KeepAlive=1	Keep Siebel portlet sessions active even if the session is idle longer than SessionTimeout. Siebel CRM is 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. In the absence of this parameter, the session timing out will lead to Siebel Open UI displaying a login dialog box in the portlet. This behavior might not be desirable in a Siebel portlet. It is recommended that you set this argument to keep the session active. For more information about the KeepAlive parameter, see *Configuring the Portlet Session to Stay Alive*.
&PtId=my_theme"	You can style a portlet application in such a way that the look and feel of the exposed application match that of the portal. The iFrame itself can be styled using a Cascading Style Sheet. For more information, see *Configuring the Use of Cascading Style Sheets Instead of iFrame Attributes*. In addition, the Siebel application can be styled according to a theme. A theme can be defined in the Siebel manifest, and the PtId argument can be used to reference the theme. The theme defined will be applied to the exposed application.
SWECmd=ExecuteLogin &SWEUserName=user_name &SWEPassword=my_password	Provide user name and password authentication arguments. ExecuteLogin is allowed only through HTTP POST. Passing user IDs and password in an HTTP request is not recommended due to security reasons. For more information, see *About Siebel Portlet Authentication and Security Requirements*.

Siebel Open UI supports HTTP POST and exposes the Siebel portlet for HTTP POST requests. The Siebel portal can send the following URL with the listed form fields:

```
http://server_name.example.com/siebel/app/callcenter/enu?
SWECmd=ExecuteLogin
SWEUserName=user_name
SWEPassword=my_password
SWEAC=SWECmd=GetApplet
SWEApplet=Quote+List+Applet
IsPortlet =1
KeepAlive=1
PtId=my_theme"
```

About Siebel Portlet Authentication and Security Requirements

Siebel Open UI portlets must be configured differently depending on whether the application is hosted in HTTP and in HTTPS. The recommended configuration guidelines are as follows:

- **HTTP.** Implement SSO and access Siebel CRM over HTTP or HTTPS, depending on the requirement.
- **HTTPS.** Implement SSO and enable SSL for Siebel CRM.

CAUTION: You should never pass user IDs and passwords in the HTTP request to a Siebel portlet. Passing user IDs and passwords exposes authentication details to the end user.

Configuring Views to Be Embedded in a Portlet

You can allow a view to be embedded in a portlet. Doing so runs the Siebel application in the portlet and navigates to a specified view. The view specified must be accessible anonymously or by user who is logged in to the Siebel Open UI client.

To allow a view to be embedded in a portlet, include the following command in the URL:

```
SWECmd=GotoView; SWEView=<View Name>; ]
```

The full URL should use the conventions in the following example:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=<View Name>
```

For example, with the Opportunities List View embedded in a portlet, the URL would use the conventions in the following URL:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=Opportunities+List+View
```

Configuring Standalone Applets to Be Embedded in a Portlet

Siebel Open UI supports standalone applets. You can expose standalone applets in a portlet. This can be achieved by providing the following `GetApplet` command in the URL:

```
SWECmd=GetApplet; SWEApplet=<Standalone Applet Name>; SWESM=<Applet's Show Mode>
```

About the SWESM Parameter

The `SWESM` parameter is the default mode for the applet to be shown, but can be changed to any one of the preconfigured modes of the applet, such as:

- Base
- Edit / Edit List
- Query

The full URL should use the conventions in the following example:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GetApplet&SWEApplet=Opportunity+List+Applet&SWESM=Base
```

About Search Specifications

When using standalone applets in portlets, the data displayed in the standalone applet can be controlled by using search specifications. The search specifications are applied to various Business Component fields on which the standalone applet is deployed. You can control the search specifications using the following parameters:

- **BCField** *<n>* **.** Defines the business component field on which to query.
- **BCFieldValue** *<n>* **.** Defines the value that the `BCField<n>` must match for the record to be displayed.
- **PBCField** *<n>* **.** Defines the parent business component field on which to query.
- **PBCFieldValue** *<n>* **.** Defines the value that the `PBCField<n>` must match for the record to be displayed.

For example, if you wanted to specify the Opportunities List applet embedded in a portlet and limit the records displayed to Opportunity Names that match "Test Opportunity" you could use the following URL:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GetApplet&SWEApplet=Opportunity+List+Applet&SWESM=Base&BCField0=Opportun
ity+Name&BCFieldValue0=Test+Opportunity
```

Search Specifications Guidelines

Adhere to these additional guidelines when defining your search specifications:

- When specifying multiple business component fields or parent business component fields, use the AND operator at the end of the final expression. Only records that satisfy all of the matching criteria are returned by the search.
- Field values can contain any type of data that is accepted by the Siebel search specification system. For example, "PBCFieldValue2=Opportunity1+OR+Opportunity2" is a valid value.
- Field values not exposed in the applet itself can still be used by the URL. These fields will be explicitly activated and used for the query.
- Search specifications applied to a URL will work in the context. Therefore, the user will not be able to access the super-set of records, unless the user navigates to the view in question.
- If a parent business component field and parent business component field value is configured in a URL, and the business component does not have a parent business component, then the specification is ignored.
- If a business component field is used in the URL that does not exist on the business component, then the URL is considered invalid and the applet will fail to build. This results in unpredictable behavior in the portlet.

Configuring View-Based Applets to Be Embedded in a Portlet

When an applet has been configured part of a view rather than as a standalone applet, it can still be exposed in a portlet. To do this, use the `GotoView` command with the following additional parameters:

```
SWECmd=GotoView; SWEView=<View_Name>; SWEApplet=<Applet_Name>
```

Only the applet specified in the portlet will be embedded in the portlet. For example, only the Opportunity List Applet will be shown using the following URL:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=Opportunity+List+View&SWEApplet=Opportunity+List+Applet
```

Note: If an applet that does not exist in the view is specified, then the URL is considered invalid and the applet fails to build. This results in unpredictable behavior in the portlet.

Configuring Multiple Siebel Portlets in Portal Applications

Siebel CRM supports exposing multiple Siebel Portlets to the Portal application in the same portlet session. By default, multiple Siebel Portlets are not exposed to the Portal application. Portal applications having one Siebel portlet works without any change. Each Siebel portlet creates a new Siebel session.

Note: It is recommended that administrators should consider the business requirements of their organizations before enabling multiple portlets and should limit the Maximum Possible Tabbed Sessions field value to minimize any adverse effect on scalability. Configuring an additional portlet than the value configured for the Maximum Possible Tabbed Sessions field leads to corruption of allowed portlets, so make sure that the number of portlets configured is always less than or equal to the Maximum Possible Tabbed Sessions field value.

To configure multiple Siebel Portlets in the Portal application, perform the following tasks:

1. *Setting Server Parameter to Enable Multiple Portlets*
2. *Configuring Application Interface Profile Parameter to Limit the Number of Portlets*
3. *Append the New PortletId Parameter to the Siebel Portlet URL*

Setting Server Parameter to Enable Multiple Portlets

To set the server parameter to enable multiple portlets, perform the following steps:

1. Log in to a Siebel client with administrative privileges.
2. Navigate to the Administration - Server Configuration screen, and then the Servers view.
3. In the Siebel Servers list, choose a Siebel Server.
4. Click the Components view tab.
5. In the Components list, select the required Application Object Manager. For example, Call Center Object Manager (ENU).
6. Select Parameters from the drop-down list before the Component Event Configuration section and click the Hidden button.
7. In the Parameter field, perform a case-sensitive search for the EnableMultiTab parameter.
8. In the Value on Restart field, enter True.
9. Restart the Siebel server.

Configuring Application Interface Profile Parameter to Limit the Number of Portlets

To set the Application Interface profile parameter to limit the number of portlets, perform the following steps:

1. Log in to Siebel Management Console.
2. Select the required Application Interface profile.
3. Select the Applications tab and expand the Enhanced Authentication section of the selected application.
4. In the Maximum Possible Tabbed Sessions field, enter a value to limit the number of browser tabs. For example, 2, if you want to allow only two browser tabs. By default, the value for Maximum Possible Tabbed Sessions field is set to 1.

   Note: This parameter is effective only when the EnableMultiTab server parameter is set to *True* for the specified Application Object Manager.

5. Save the profile.

Append the New PortletId Parameter to the Siebel Portlet URL

The new PortletId parameter is required in the Siebel portlet URL when more than one portlet needs to be added in the portal application. PortletId should be a valid numeric number. The following sample code snippet illustrates two Siebel portlets exposed in a portal application.

```
<HTML>
  <BODY>
     <IFRAME src = "https://server_address/siebel/app/application/lang?SWECmd=GotoView&IsPortlet
=1&PortletId=1&other_arguments"
          style="height:500px;width:1000px"> </IFRAME>
      <IFRAME src = “https://server_address/siebel/app/application/lang?SWECmd=GetApplet&IsPortlet
=1&PortletId=2&other_arguments"
           style="height:500px;width:1000px"> </IFRAME>
    </BODY>
</HTML>
```

Configuring Advanced Options

This topic describes advanced options when configuring Siebel Open UI in an external application.

Configuring Multiple Command Chaining in a URL

Use the `SWEAC` parameter to chain more than one command in a URL. An example, where this might be useful is a situation where you want to navigate to a certain view and create a new record in that view's active applet.

To configure multiple command chaining in a URL, include the following attribute in the URL:

```
SWEAC=SWECmd=NewRecord]
```

The full URL should use the conventions in the following example:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=Opportunities+List+View&SWEAC=SWECmd=NewRecord
```

The preceding example runs the Siebel application in the portlet and takes the context to the Opportunity View to create a new record in the active applet on that view.

Configuring the Portlet Session to Stay Alive

Siebel sessions that are not in use will eventually expire. The time for which the session is kept alive is determined by the value of the SessionTimeout server parameter. In some cases when exposing Siebel CRM as a portlet expiring sessions this might not be optimal.

To override the SessionTimeout server parameter so that the portlet session stays alive, include the following attribute in the URL:

```
KeepAlive=1
```

[Other values for this parameter are as follows: `TRUE`, `T`, `ON`, and `Y`.]

The full URL should use the conventions in the following example:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=Contact+List+View&KeepAlive=1
```

When using the KeepAlive attribute, consider these additional guidelines:

- The KeepAlive attribute value is enforced by monitoring periodic client pings to the Siebel Server. Consequently, the client must be on a network connected to the server.

- If the KeepAlive attribute value is omitted or set to `FALSE` the session will eventually timeout and a login screen is returned to the portlet.
- Once the KeepAlive attribute is set to `TRUE` by a request (either the URL or a subsequent message-based communication) it cannot be changed to `FALSE` by a subsequent request.

Configuring the Use of Cascading Style Sheets Instead of iFrame

The iFrame tag supports a number of attributes, which can be used to control the visual formatting of the portlet content. For a full list of the attributes, see the following W3C website:

http://www.w3.org/wiki/HTML/Elements/iframe

In recent HTML revisions, many attributes are being deprecated. Consequently, it is recommended that cascading style sheets be used for visual formatting.

Siebel Open UI attaches CSS classes for the portlet iFrame. In Siebel Open UI, the CSS can be applied by defining a theme in the Theme.js file and passing the theme name as a parameter in the URL under PtId.

The full URL should use the conventions in the following example:

```
http://<siebel_server>/<siebel/app/application>/lang?IsPortlet
=1&SWECmd=GotoView&SWEView=Contact+List+View&KeepAlive=1&PtId=CUSTOM_PORTLET_THEME
```

Where `CUSTOM_PORTLET_THEME` is defined in Theme.js. If the argument value is omitted, invalid, or cannot be found in Theme.js, then Siebel Open UI will use the default theme.

For more information about customizing themes, see *Customizing Themes*.

Configuring Communications with Siebel Portlets When Hosted Inside iFrame

This topic outlines the Siebel Server parameter configurations that are required and optional to enable communication with Siebel portlets when hosted inside an iFrame. These parameters can be modified for the Siebel Component with which the functionality is meant to communicate. The instructions in this topic are not required when cross-domain communications are not needed.

Planning Across Domain Integrations

Siebel Open UI can be used in IFrames in the same domain and supports use across domains. The following settings support this feature:

- **Xframe-options allow-from.** Allows Siebel CRM to be hosted inside a portal. The portal application name has to be listed as the Allow-From value.
- **PortletOriginList.** Gives the list of allowed applications to communicate with Siebel Open UI when it is hosted inside an iFrame.

Planning Cross-Domain Integrations

You can use the X-Frame-Options HTTP header to determine whether or not Siebel Open UI can display a page in a browser in a frame or in an iFrame. This capability is useful to avoid a potential security problem by making sure a hacker cannot embed the content that Siebel Open UI provides into another application. The XFrameOptions parameter

is a hidden Siebel Server parameter that you can use to control the value of the X-Frame-Options header. You can set it to one of the following values:

- **SAMEORIGIN.** Display the page only in a frame that resides in the same location as the page.This is the default value.
- **ALLOW-FROM.** *url* Display the page only in a frame that resides in the specified location. If an external application accesses a Siebel URI, then you specify the URI that this external application uses. For example, if the external application uses my_url.com, then you use the following value: ALLOW-FROM http://my_url.com/ If a browser (such as Chrome or Safari) does not support ALLOW-FROM, then the browser ignores it.
- **DENY.** Do not display the page in a frame or in an iFrame.

Make sure that HTTPS/HTTP transports match for cross-domain sites.

To configure communications with Siebel portlets when hosted inside an iFrame

1. Set up the Siebel Server parameters:
   a. Log in to a 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	This is a required parameter. It is a unique key configured as a server parameter. The source portal program must pass this key to call the Siebel application exposed as the portlet. The messaging object used to communicate with Siebel Portal will need to contain a parameter msg.Key. The msg.Key must match the key configured in this parameter. If the messaging object does not contain a key, or contains an invalid one, the invocation will result in an error in the Siebel portlet.
PortletOriginList	This is a required parameter. It defines the list of valid domains from which the Siebel portlet will accept a communication request. A comma separated list can be provided for this parameter. Any invocations coming from domains that are not listed here will cause an error in the Siebel portlet.
PortletMaxAllowedAttempts	This is an optional parameter. Its default value is 3. This parameter specifies the number of unsuccessful communication attempts with the portlet before Siebel Open UI blocks any subsequent calls. An unsuccessful call can occur in the following situations: - A domain attempts to send a communication request to the portlet, but the PortletOriginList does not specify this domain. - The portlet_key sent by the communicating domain does not match the parameter specified in the Siebel server. The Siebel portal will remain blocked up to the time extent as defined by PortletBlockedInterval after which Siebel Open UI resets the unsuccessful attempts to zero
PortletBlockedInterval	This is an optional parameter. Its default value is 900 seconds. This parameter specifies the time in seconds for which Siebel portlet will remain blocked to any

Parameter	Description
	communication attempt from the hosting portal or a neighboring portlet after having exceeded the number of unsuccessful communication attempts (as defined by PortletMaxAllowedAttempts). During this time, the Siebel portlet will still be open to access by the user of the application. However, no programmatic access is permitted.

2. Based on your configuration, the portal, or another portlet in the portal, 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.
   - *portlet_key* must specify the value that you specify for the PortletAPIKey server parameter in Step . 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 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 .

```
document.getElementById('siebelframeid').contentWindow.postMessage(msg, '*');
```

   This code invokes a change in the Siebel Portlet window, so that the requested view or applet will get loaded in the content area.
4. You can use several SWE commands to display a Siebel portlet in Siebel Open UI. For security reasons, you can use only the GotoView and GetApplet method to call a Siebel portlet from an external application. GotoPage

and GotoPageTab are not applicable to Siebel Open UI. You can use the commands in the following table within a Siebel portlet. You cannot use them to call a portlet.

Supported Values	Inside External Siebel Application	Called from UI Element Inside Siebel Portlet Container	Called from Outside Siebel Portlet Container
CanInvokeMethod	Yes	Yes	No
ExecuteLogin	Yes. It is not supported for HTTP GET. It is supported through HTTP POST.	Not applicable for this use case.	No. Yes. It is not supported for HTTP GET. It is supported through HTTP POST.
GotoView	Yes. Use only when invoked from the browser address bar by refresh or history navigation.	Yes	Yes
GetApplet	Yes	Yes	Yes
InvokeMethod	Yes	Yes	No For more information, see *Allowing Blocked Methods for HTTP GET Access*.
LoadService	Yes	Yes	No
Login	Yes	Not applicable to Siebel Open UI.	Not applicable (use SSO or similar)
Logoff	Yes	Not applicable to Siebel Open UI.	No
ReloadCT	Yes	Yes	No

Additional Considerations

The following list outlines additional considerations when displaying data from Siebel Open UI in external applications:

- All parameters passed in a URL need to be URL-encoded. For example, "Account List View" would become "Account+List+View" or "Account%20List%20View". For more information on URL encoding, refer to:

  http://en.wikipedia.org/wiki/Percent-encoding
- Anonymous sessions are supported in portlet expositions.
- Tasks Workflow URLs are also supported in portlets.
- SWE Commands are limited to the ones mentioned in Step 4 of *Configuring Communications with Siebel Portlets When Hosted Inside iFrame*. However, other parameters may be passed in portlet mode to the Siebel server. They will be honored by the server depending on the context.
- If the content in the Siebel portlet is bootstrapped to load an applet using the GetApplet method, then the subsequent messaging to the portlet will be limited to whether the applet can be invoked. Operations such as invoking of popups or navigating to other views will not be supported. If these are required, the portlet must be bootstrapped via the GotoView call. For more information, see *Configuring Standalone Applets to Be Embedded in a Portlet*.

Limitations

The following list outlines limitations when displaying data from Siebel Open UI in external applications:

- Siebel CRM supports only one portlet in a valid Siebel session. Consuming more than one portlet that is targeted to same Siebel session is not supported.
- Opening Siebel Open UI in multiple browser tabs that share the same Siebel session ID is not supported.
- Portal communications as described in *Configuring Communications with Siebel Portlets When Hosted Inside iFrame*, is not supported in any version of Microsoft Internet Explorer. Siebel Open UI uses HTML 5 specified Cross Document Messaging, that is not fully supported in the latest version of Internet Explorer.

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.

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.linked.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.linked.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.example.com/siebel/app/callcenter/enu?
      SWEUserName=user_name&SWEPassword=user_password&SWECmd=ExecuteLogin&S
      WEAC=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/siebel/app/callcenter/enu?
SWEUserName=%48%4B%49%4D&SWEPassword=%48%4B%49%4D&SWECmd=ExecuteLogin
&SWEAC=SWECmd=GotoView&SWEView=Opportunity+List+View&IsPortlet=1&SWEApplet=Oppo
rtunity+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
```

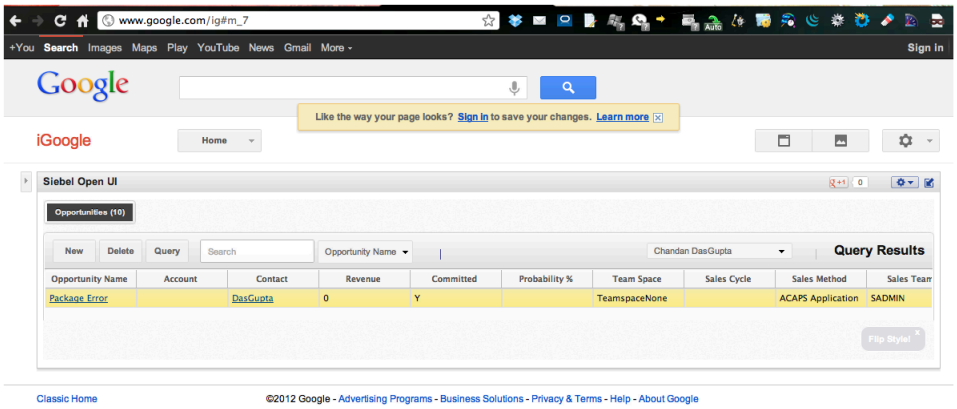
It is strongly recommended that you use Web Single Sign-On (SSO) to handle this user name and password authentication. For more information, see the topic that describes the URL Login in *Siebel Security Guide* .

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 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 that shown in the following image.



SWE API

This topic contains reference information about SWE commands, methods, and arguments. The command is described in the following section.

SWE Commands Available in Siebel Open UI

You can use several SWE commands to display a Siebel portlet in the external application. For security reasons, you can use only the GotoView and GetApplet methods to call a Siebel portlet from an external application. GotoPage and

GotoPageTab are not applicable in Siebel Open UI. You can use the commands listed in the following table within a Siebel portlet. You cannot use them to call a portlet.

Supported Values	Inside Siebel Application	Called from UI Element Inside Siebel Portlet Container	Called from Outside Siebel Portlet Container
CanInvokeMethod	Yes	Yes	No
ExecuteLogin	Yes This is not supported for HTTP GET. It is supported through HTTP POST.	Not applicable for this use case.	Yes This is not supported for HTTP GET. It is supported through HTTP POST.
GotoView	Yes Use only when invoked from the browser address bar by refresh or history navigation.	Yes	Yes
GetAplet	Yes	Yes	Yes
InvokeMethod	Yes	Yes	No
LoadService	Yes	Yes	No
Login	Yes	Not applicable.	Not applicable (use SSO or similar).
Logoff	Yes	Not applicable.	No
ReloadCT	Yes	Yes	No

Web Engine HTTP TXN Business Service

This chapter describes the Web Engine HTTP TXN Business Service. It contains the following information:

- *About the Web Engine HTTP TXN Business Service*
- *Web Engine HTTP TXN Business Service API*
- *Example of Using Web Engine HTTP TXN Business Service*
- *Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service*

About the Web Engine HTTP TXN Business Service

HTTP provides several means to allow Web servers to obtain information from the browser. The most familiar example is when a user enters data into a form on a Web page and the data is sent to the Web server, which can access the value of each form field. This example illustrates sending form field parameters to the Web server with a POST method. In general, a browser can send cookies, headers, query string parameters, and form field parameters to the Web server. Web servers can also respond to the browser with cookies and custom headers. The Web Engine HTTP TXN Business Service allows Siebel Business Applications to retrieve or set cookies, headers, and query string and form field parameters.

The Web Engine HTTP TXN Business Service can be invoked by scripts or by workflow. The inbound HTTP request to the Siebel Web Engine (SWE) is parsed and the business service returns property sets containing cookies, headers, or parameters. In addition, server variables, which are not a part of the HTTP request header, can also be retrieved. The business service can also set a custom cookie or header in the HTTP response header generated by the SWE. The business service gives complete control over the request header received and the response header sent by the SWE.

For more information, see the following topics:

- *Web Engine HTTP TXN Business Service API*
- *Example of Using Web Engine HTTP TXN Business Service*
- *Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service*

Web Engine HTTP TXN Business Service API

The following table lists the methods exposed by the Web Engine HTTP TXN Business Service.

Method	Description	Parameters
GetAllRequestCookies	Retrieves all request cookies sent from the client to the server.	**InputArguments:** Ignored. **OutputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetAllRequestHeaders	Retrieves all request headers sent from the client to the server.	**InputArguments:** Ignored. **OutputArguments**: Property Set containing the HTTP Parameter name-value pairs.
GetAllRequestParameters	Retrieves all request parameters sent from the client to the server.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the HTTP Parameter name-value pairs.
GetAllResponseCookies	Retrieves all response cookies sent from the server to the client.	**InputArguments:** Ignored. **OutputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.

Method	Description	Parameters
GetAllResponseHeaders	Retrieves all response headers sent from the server to the client.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the HTTP Header name-value pairs.
GetAllServerVariables	Retrieves all server variables.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the Server Variable name-value pairs.
GetClientCertificate	Retrieves the client certificate info.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing certificate name-value pairs. Currently only returns Common Name (CN) property of the certificate.
GetRequestCookies	Retrieves the request cookies named in InputArguments.	**InputArguments:** Property Set containing the cookie names to retrieve. **OutputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetRequestHeaders	Retrieves the request headers named in InputArguments.	**InputArguments:** Property Set containing the header names to retrieve. **OutputArguments:** Property Set containing the HTTP Header name-value pairs.
GetRequestInfo	Retrieves the request Web Session, Headers, Cookies, Parameters and Client Certificate information in one call.	**InputArguments:** Ignored **OutputArguments:** Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers', 'Cookies', 'Parameters' or 'ClientCertificate'. The Web Session information is simply stored as properties of OutputArguments.
GetRequestParameters	Retrieves the request parameters named in InputArguments.	**InputArguments:** Property Set containing the parameter names to retrieve. **OutputArguments:** Property Set containing the HTTP Parameter name-value pairs.
GetResponseCookies	Retrieves the response cookies named in InputArguments.	**InputArguments:** Property Set containing the cookie names to retrieve. **OutputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetResponseHeaders	Retrieves the response headers named in InputArguments.	**InputArguments:** Property Set containing the header names to retrieve.

Method	Description	Parameters
		OutputArguments: Property Set containing the HTTP Header name-value pairs.
GetResponseInfo	Retrieves the response Headers and Cookies in one call.	**InputArguments:** Ignored. **OutputArguments:** Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers' or 'Cookies'. Content Type and Status are simply stored as properties of OutputArguments.
GetServerVariables	Retrieves the server variables named in InputArguments.	**InputArguments:** Property Set containing the server variable names to retrieve. **OutputArguments:** Property Set containing the Server Variable name-value pairs.
GetWebSessionInfo	Retrieves the client's Web session information.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the Web session name-value pairs—SessionName; Cookie Name; SessionId; Web Session ID; SessionFrom (Value is 'URL' or 'COOKIE').
SetResponseCookies	Sets the response cookies to the values in InputArguments.	**InputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name. The PERSISTENT property determines whether the cookie persists between sessions. If the value is Y, then the cookie persists between browser sessions. Otherwise, the cookie exists for one session at a time. **OutputArguments:** Ignored.
SetResponseHeaders	Sets the response headers to the values in InputArguments.	**InputArguments:** Property Set containing the HTTP Header name-value pairs. **OutputArguments:** Ignored.
SetResponseInfo	Sets the response Headers and Cookies in one call.	**InputArguments:** Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers' or 'Cookies'. Content Type and Status are simply stored as properties of InputArguments. **OutputArguments:** Ignored.

Example of Using Web Engine HTTP TXN Business Service

To invoke each method of the Web Engine HTTP TXN Business Service and write the results to a text file, use the following procedures:

- *Adding Sample Code for Displaying Results of Using the Business Service*
- *Adding Sample Code for Invoking Methods of the Business Service*

Adding Sample Code for Displaying Results of Using the Business Service

The following procedure shows how to add sample code for displaying results of the Web Engine HTTP TXN Business Service.

To add sample code for displaying results of Web Engine HTTP TXN Business Service

1. In Oracle's Siebel Tools, navigate to the desired Applet object, in the Object Explorer.
2. Lock the project, if required.
3. Right-click and select the Edit Server Script option.
4. Add the following three functions, individually to the declarations section:
    - WebApplet_OutputChildPropertySets
    - WebApplet_OutputProperties
    - WebApplet_OutputPropertySet

Sample Code Functions

Sample code for the WebApplet_OutputChildPropertySets Function:

```
function WebApplet_OutputChildPropertySets(oPropertySet, nLevel, fp)
{
var oChildPropSet;
var nChild = 0;
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('CHILD PROPERTY SETS\n',fp);
Clib.fputs('-------------------------------------\n',fp);
if ( oPropertySet.GetChildCount() == 0 )
{
Clib.fputs('(NONE)\n',fp);
}
else
{
for ( nChild = 0; ( nChild <= oPropertySet.GetChildCount() - 1 ) ; nChild++ )
{
oChildPropSet = oPropertySet.GetChild(nChild);
WebApplet_OutputPropertySet (oChildPropSet, nLevel+1, fp);
}
}
}
```

Sample code for the WebApplet_OutputProperties Function:

```
function WebApplet_OutputProperties(oPropertySet, nLevel , fp )
{
var strName;
var strValue;
Clib.fputs('-------------------------------------\n',fp);
```

```
Clib.fputs('PROPERTIES\n',fp);
Clib.fputs('-------------------------------------\n',fp);
if (oPropertySet.GetPropertyCount() == 0 )
{
Clib.fputs('(NONE)\n',fp);
}
else
{
strName = oPropertySet.GetFirstProperty();
while ( strName != '')
{
Clib.fputs(strName + ' : ' + oPropertySet.GetProperty(strName) + '\n' ,fp);
strName = oPropertySet.GetNextProperty();
}
}
}
```

Sample code for the WebApplet_OutputPropertySet Function:

```
function WebApplet_OutputPropertySet(oPropertySet, nLevel, fp )
{
Clib.fputs('\n',fp);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('START' + ' ',fp);
Clib.fputs('LEVEL : ' + nLevel + '\n', fp);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('TYPE : ' + oPropertySet.GetType() + '\n',fp);
Clib.fputs('VALUE : ' + oPropertySet.GetValue() + '\n',fp);
WebApplet_OutputProperties(oPropertySet, nLevel, fp);
WebApplet_OutputChildPropertySets(oPropertySet, nLevel, fp);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('END' + ' ',fp);
Clib.fputs('LEVEL : ' + nLevel + '\n',fp);
Clib.fputs('-------------------------------------\n',fp);
}
```

Adding Sample Code for Invoking Methods of the Business Service

The following procedure shows how to add sample code for invoking methods of the Web Engine HTTP TXN Business Service.

To add sample code for invoking methods of Web Engine HTTP TXN Business Service

1. Add the code from *Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service* to the WebApplet_InvokeMethod event.
2. Compile the project.
3. Start the Siebel application.
4. Navigate to the applet where the server script has been placed.
5. Perform an action on the applet that invokes a SWE method (for example, change the record or create a new record).

   The code generates a text file in the `bin` directory where the Siebel application is installed containing results of each method of the Web Engine HTTP TXN Business Service.

Sample Output

The following is an excerpt of the resulting text file.

```
====================================
WebApplet InvokeMethod event:
====================================
====================================
```

```
Method: GetAllRequestCookies
=====================================
-------------------------------------
START LEVEL : 0
-------------------------------------
TYPE : COOKIES
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
(NONE)
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
-------------------------------------
START LEVEL : 1
-------------------------------------
TYPE : SWEUAID
VALUE : 1
-------------------------------------
PROPERTIES
-------------------------------------
Max-Age : -1
Domain :
Path :
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 1
-------------------------------------
-------------------------------------
END LEVEL : 0
-------------------------------------
=====================================
Method: GetAllRequestHeaders
=====================================
-------------------------------------
START LEVEL : 0
-------------------------------------
TYPE : HEADERS
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
HOST : <host computer name>
CACHE-CONTROL : no-cache
CONNECTION : Keep-Alive
COOKIE : SWEUAID=1
USER-AGENT : Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461; SV1; .NET
CLR 1.1.4322)
CONTENT-TYPE : application/x-www-form-urlencoded
ACCEPT-ENCODING : deflate
CONTENT-LENGTH : 348
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 0
-------------------------------------
=====================================
Method: GetAllRequestParameters
=====================================
-------------------------------------
```

```
START LEVEL : 0
-------------------------------------
TYPE : PARAMETERS
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
SWEActiveView : Account List View
SWERowIds :
SWEP :
SWESP : false
SWECmd : InvokeMethod
SWEMethod : PositionOnRow
SWER : 1
SWEControlClicked : 0
SWEIgnoreCtrlShift : 0
SWEVI :
SWEActiveApplet : Account List Applet
SWERPC : 1
SWEReqRowId : 1
SWEView : Account List View
SWEC : 3
SWERowId : 1-6
SWEShiftClicked : 0
SWETS : 1118939959734
SWEApplet : Account List Applet
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 0
-------------------------------------
```

Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service

This topic contains the sample code for invoking the methods of the Web Engine HTTP TXN Business Service and writing the results to a text file. For more information, see *Example of Using Web Engine HTTP TXN Business Service*.

Add the following sample code to the WebApplet_InvokeMethod event:

```
function WebApplet_InvokeMethod (MethodName)
{
var fp = Clib.fopen('testfile.txt','a');
if ( fp == null )
{
TheApplication().RaiseErrorText(" ERROR Opening File ")
}
else
{
var oBS = TheApplication().GetService('Web Engine HTTP TXN');
var Inputs = TheApplication().NewPropertySet();
var Outputs = TheApplication().NewPropertySet();
var Headers = TheApplication().NewPropertySet();
var Cookies = TheApplication().NewPropertySet();
var tmpCookie = TheApplication().NewPropertySet();

Clib.fputs('====================================\n',fp);
Clib.fputs('WebApplet InvokeMethod event:\n',fp);
Clib.fputs('====================================\n',fp);
```

```
Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllRequestCookies\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ( 'GetAllRequestCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllRequestHeaders\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllRequestHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllRequestParameters\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllRequestParameters', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllResponseCookies\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllResponseCookies', Inputs, Outputs)
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllResponseHeaders\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllResponseHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetAllServerVariables\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllServerVariables', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetRequestCookies\n',fp);
Clib.fputs('=====================================\n',fp);
```

```
Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MY-COOKIE', '');
Inputs.SetProperty ('TestCookie', '');
Inputs.SetProperty ('Test1Cookie', '');

oBS.InvokeMethod ('GetRequestCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetRequestHeaders\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MyHEADER', '');
Inputs.SetProperty ('MY_TEST', '');
Inputs.SetProperty ('CONTENT-TYPE', '');
Inputs.SetProperty ('CONTENT-LENGTH', '');

oBS.InvokeMethod ('GetRequestHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetRequestInfo\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetRequestInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetRequestParameters\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('TestQstr', '');
Inputs.SetProperty ('SWEActiveView', '');
Inputs.SetProperty ('SWECmd', '');
Inputs.SetProperty ('SWEMethod', '');
Inputs.SetProperty ('TestParam', '');

oBS.InvokeMethod ('GetRequestParameters', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetResponseCookies\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('My-Test-COOKIE', '');
Inputs.SetProperty ('_sn', '');
oBS.InvokeMethod ('GetResponseCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);
```

```
Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetResponseHeaders\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('Content-Language', '');
Inputs.SetProperty ('MyHeader', '');

oBS.InvokeMethod ('GetResponseHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetResponseInfo\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetResponseInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetServerVariables\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('AUTH-USER-ID', '');
Inputs.SetProperty ('SERVER-NAME', '');

oBS.InvokeMethod ('GetServerVariables', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: GetWebSessionInfo\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetWebSessionInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: SetResponseCookies\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('My_Test_Cookie');
tmpCookie.SetValue ('Cookie Value for My_Test_Cookie');
tmpCookie.SetProperty ('Max-Age', '23434343');
tmpCookie.SetProperty ('Domain', '.example.com');
```

```
tmpCookie.SetProperty ( 'Path', 'eapps/test/cookie/path');

Inputs.AddChild (tmpCookie);

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('Another_Cookie');
tmpCookie.SetValue ('Cookie Value for Another_Cookie');
tmpCookie.SetProperty ('Max-Age', '23434343');
tmpCookie.SetProperty ('Domain', 'esales.example.com');
tmpCookie.SetProperty ('Path', 'esales/cookie/path');

Inputs.AddChild (tmpCookie);

oBS.InvokeMethod ('SetResponseCookies', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Input Cookies\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp);

oBS.InvokeMethod ('GetAllResponseCookies', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Output Cookies\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: SetResponseHeaders\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MyHeader', 'THIS is MyHeader');

oBS.InvokeMethod ('SetResponseHeaders', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Input Headers\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp)

oBS.InvokeMethod ('GetAllResponseHeaders', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Output Headers\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('=====================================\n',fp);
Clib.fputs('Method: SetResponseInfo\n',fp);
Clib.fputs('=====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
Headers.Reset();
Cookies.Reset();

Headers.SetType ('HEADERS');
Headers.SetProperty ('ABC_RESPONSE_HEADER1', 'RESPONSE_HEADER1 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER2', 'RESPONSE_HEADER2 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER3', 'RESPONSE_HEADER3 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER4', 'RESPONSE_HEADER4 Value');
Inputs.AddChild( Headers);
```

```
Cookies.SetType('COOKIES');

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('My_Test_Cookie2');
tmpCookie.SetValue ( 'Cookie Value for My_Test_Cookie2');
tmpCookie.SetProperty ( 'Max-Age', '23434343');

Cookies.AddChild (tmpCookie);

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('Another_Cookie2');
tmpCookie.SetValue ('Cookie Value for Another_Cookie2');
tmpCookie.SetProperty ('Max-Age', '23434343');

Cookies.AddChild (tmpCookie);

Inputs.AddChild (Cookies);

oBS.InvokeMethod ('SetResponseInfo', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Input Info\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp);

oBS.InvokeMethod ('GetResponseInfo', Inputs, Outputs);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('Output Info\n',fp);
Clib.fputs('-------------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fclose(fp);
}
}
```

8 Customizing Siebel Open UI for Siebel Mobile Disconnected

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*
- *Doing General Customization Tasks for Siebel Mobile Disconnected*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Customizing Siebel Service for Siebel Mobile Disconnected Clients*
- *Methods You Can Use to Customize Siebel Mobile Disconnected*

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*
- *Operations You Cannot Customize When Clients Are Offline*
- *Process of Customizing Siebel Open UI for Siebel Mobile Disconnected*

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. Some 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 run-time 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*.

- 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 the 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* and *Customizing Siebel Service for Siebel Mobile Disconnected Clients*.

To customize Siebel Open Ulfor Siebel Mobile Disconnected

1. Configure the manifest, if necessary.

   For more information, see *Modifying Manifest Files for Siebel Mobile Disconnected*.
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*.
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*.
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*.
5. Test your modifications.

Doing General Customization Tasks for Siebel Mobile Disconnected

This topic describes how to do general customization tasks for Siebel Mobile Disconnectedin Siebel Open UI. It includes the following topics:

- *Modifying Manifest Files for Siebel Mobile Disconnected*
- *Registering Methods to Make Sure Siebel Open UI Runs Them in the Correct Sequence*
- *Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects*
- *Using Custom JavaScript Methods*
- *Using Custom Siebel Business Services*
- *Configuring Data Filters*
- *Configuring Objects That Siebel Open UI Does Not Display in Clients*
- *Configuring Error Messages for Disconnected Clients*
- *About Siebel Mobile Application Logging*

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:

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\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.

For more information about the **language_code**, see *Languages That Siebel Open UI Supports*.

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
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 (if doing this task on the Siebel Server):

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\language_code
   ```

4. Enter the following command:

   ```
   Perl mergemanifest.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 mergemanifest.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.

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*.
2. Locate the code that includes the business service call that you must modify.
3. 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*, *FirstRecord Method* and *ExecuteQuery Method*.
     In this example, you replace the code that you located in Step 2 with the following code:

```
PharmaCallSubmitsvc.prototype.Submit = function () {
```

```
var currRetValue={err:false}, retObj;
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
currRetValue={err:false,retVal:bRet};
}
}
else{
SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("messageKey", errParamArray);
currRetValue={err:true};
}
return currRetValue;
};
```

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*.
   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 (SiebelApp.pharmacallsvc) === "undefined") {
       SiebelJS.Namespace('SiebelApp.pharmacallsvc');
      ```
   b. Define the variables:
      ```
      var oconsts = SiebelApp.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("DOUIREG_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("DOUIREG_OBJ_TYPE")] =
oconsts.get("DOUIREG_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("DOUIREG_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("DOUIREG_SRVC_NAME")] = "pharmacallsvc";
```

g. Specify the name of the custom service method that contains the customization of the WriteRecord method:

```
inputObj [oconsts.get("DOUIREG_SRVC_MTDH")] = "WriteRecord";
```

h. Specify the type of customization:

```
inputObj [oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
```

3. Register the custom JavaScript method that you specified in Step 2. This code calls the ServiceRegistry method:

```
SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
```

4. Define the constructor:

```
SiebelApp.pharmacallsvc = (function () {
function pharmacallsvc() {
}
```

5. Extend the custom JavaScript class:

```
SiebelJS.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") {
 SiebelJS.Namespace('SiebelApp.pharmacallsvc');
 var oconsts = SiebelApp.Offlineconstants;
 var inputObj = {};
```

```
 inputObj [oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Professional Call - Mobile";
 inputObj [oconsts.get("DOUIREG_OBJ_TYPE")] =
oconsts.get("DOUIREG_OBJ_TYPEBUSCOMP");
 inputObj [oconsts.get("DOUIREG_OBJ_MTHD")] = "WriteRecord";
 inputObj [oconsts.get("DOUIREG_SRVC_NAME")] = "pharmacallsvc";
 inputObj [oconsts.get("DOUIREG_SRVC_MTDH")] = "WriteRecord";
 inputObj [oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
 SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
 SiebelApp.pharmacallsvc = (function () {
 function pharmacallsvc() {
 }
 SiebelJS.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;
 } ());
}
```

7. If you want Siebel Open UI to anonymously register existing applet and business component objects you can use anonymous registration. This allows administrators to have a common customization across all applets or all business components.

   For example, in order to have the ability to print or click on a specific button in any applet, the following registration will give the handle of invoke a method in any applet, because the ObjectName is deliberately omitted:

```
inputArgs[oconsts.get("DOUIREG_OBJ_NAME")] = "";
inputArgs[oconsts.get("DOUIREG_OBJ_TYPE")]=oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
inputArgs[oconsts.get("DOUIREG_OBJ_MTHD")] = "InvokeMethod";
inputArgs[oconsts.get("DOUIREG_SRVC_NAME")] = "CustomDMService";
inputArgs[oconsts.get("DOUIREG_SRVC_MTDH")] = "InvokeMethodPrint";
inputArgs[oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
```

   In this case, InvokeMethodPrint will be called for all applets as PRE whenever InvokeMethod is called for any applet.

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*. 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 through Step 6 in the topic *Customizing Predefined Business Components* with the following differences:
  - For Step 2, Step c, in the topic *Customizing Predefined Business Components* specify the applet where Siebel Open UI applies your customization. In this example, you specify the Pharma Call Entry Mobile applet:

    ```
    inputObj [oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Call Entry Mobile";
    ```

  - For Step 2, Step d, in the topic *Customizing Predefined Business Components* specify the type of object that you are customizing. You specify an applet instead of a business component:

    ```
    inputObj [oconsts.get("DOUIREG_OBJ_TYPE")] =
    ```

```
oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
```

The following code is the completed code for this topic:

```
if (typeof (SiebelApp.pharmacallsvc) === "undefined") {
 SiebelJS.Namespace('SiebelApp.pharmacallsvc');
 var oconsts = SiebelApp.Offlineconstants;
 var inputObj = {};
 inputObj [oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Call Entry Mobile";
 inputObj [oconsts.get("DOUIREG_OBJ_TYPE")] =
oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
 inputObj [oconsts.get("DOUIREG_OBJ_MTHD")] = "InvokeMethod";
 inputObj [oconsts.get("DOUIREG_SRVC_NAME")] = "pharmacallsvc";
 inputObj [oconsts.get("DOUIREG_SRVC_MTDH")] = "InvokeMethod";
 inputObj [oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
 SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
 SiebelApp.pharmacallsvc = (function () {
 function pharmacallsvc() {
 }
 SiebelJS.Extend(pharmacallsvc, SiebelApp.ServiceModel);
 pharmacallsvc.prototype.InvokeMethod = function (psInputArgs) {//get the inputs
 var psOutArgs = SiebelApp.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 (SiebelApp.pharmacallsvc) === "undefined") {
 SiebelJS.Namespace('SiebelApp.pharmacallsvc');
 var inputObj = {};
 var oconsts = SiebelApp.Offlineconstants;
 inputObj[oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Call Entry Mobile";
 inputObj[oconsts.get("DOUIREG_OBJ_TYPE")] =
oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
 inputObj[oconsts.get("DOUIREG_OBJ_MTHD")] = "CanInvokeMethod";
 inputObj[oconsts.get("DOUIREG_SRVC_NAME")] = "pharmacallsvc";
 inputObj[oconsts.get("DOUIREG_SRVC_MTDH")] = "CanInvokeMethod";
 inputObj[oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
 SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
 inputObj[oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Call Entry Mobile";
```

```
 inputObj[oconsts.get("DOUIREG_OBJ_TYPE")] =
oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
 inputObj[oconsts.get("DOUIREG_OBJ_MTHD")] = "InvokeMethod";
 inputObj[oconsts.get("DOUIREG_SRVC_NAME")] = "pharmacallsvc";
 inputObj[oconsts.get("DOUIREG_SRVC_MTDH")] = "InvokeMethod";
 inputObj[oconsts.get("DOUIREG_EXT_TYPE")] = oconsts.get("DOUIREG_EXT_TYPEPRE");
 SiebelApp.S_App.GetModel().ServiceRegistry(inputObj);
 SiebelApp.pharmacallsvc = (function () {
 function pharmacallsvc(pm) {
 }
 SiebelJS.Extend(pharmacallsvc, SiebelApp.ServiceModel); //Extending
 pharmacallsvc.prototype.InvokeMethod = function (psInputArgs) {
 var svcMthdName = "";
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
```

For more information about this code, see the description about the inputObj argument in *ServiceRegistry Method*. Also see *CanInvokeMethod Method* and *Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects*.

3. Get the value of the MethodName argument from the psInputArgs method:

```
svcMthdName = psInputArgs.GetProperty("MethodName").toString();
```

4. Call the Submit method:

```
if (svcMthdName === "Submit") {
 retObj=this.Submit();
```

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);
 currRetValue=({err: "", retVal: psOutArgs});
 }
 else {
 psOutArgs.SetProperty("Invoked", true);
 currRetValue=({err: retObj.err, retVal: psOutArgs});
 }
 });
return currRetValue;}
```

   - 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);
 currRetValue=({err: "", retVal: psOutArgs});
 }
 return(currRetValue);
};
```

   - 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:

```
pharmacallsvc.prototype.CanInvokeMethod = function (psInputArgs) {
 var currRetValue={err:false}, retObj;
```

```
var psOutArgs = SiebelApp.S_App.NewPropertySet();
var svcMthdName = "";
svcMthdName = psInputArgs.GetProperty("MethodName").toString();
if (svcMthdName === "Submit") {
psOutArgs.SetProperty("Invoked", true);
psOutArgs.SetProperty("RetVal", true);
currRetValue=({err: "", retVal: psOutArgs});
}
```

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);
currRetValue=({err: "", retVal: psOutArgs});
}
return(currRetValue);
};
pharmacallsvc.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("DOUIREG_OBJ_NAME")]= "business_service";
inputObj[oconsts.get("DOUIREG_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("DOUIREG_OBJ_NAME")]= "LS Pharma Validation Service";
 inputObj[oconsts.get("DOUIREG_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*
- *ServiceRegistry Method*
- *Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects*

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:

```
PharmaCallValidatorsvc.prototype.CanInvokeMethod = function (svcMthdName) {
 var currRetValue={err:false}, retObj;
 if (svcMthdName === "CallValidate") {
 currRetValue={ err: false, retVal: true };
 return currRetValue;
 }
 else {
 return SiebelApp.PharmaCallValidatorsvc.superclass.CanInvokeMethod.call(this,
svcMthdName);
 }
};
```

   For more information about the methods that this step uses, see *CanInvokeMethod Method*.

4. Depending on whether you want to make a call from service to service, or to a standalone service, use one of the following methods:

   a. To make a call from one service to another service, use InvokeMethod. This method will call your custom business service method.

      For example, the following code calls the CallValidate business service method:

```
 PharmaCallValidatorsvc.prototype.InvokeMethod = function (svcMthdName,
psinpargs) {
 var currRetValue={err:false}, retObj;
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
 if (!svcMthdName) {
 currRetValue=({err: "", retVal: true});
 return currRetValue;
 }
 if (svcMthdName === "CallValidate") {
 retObj=this.CallValidate(psinpargs);
 psOutArgs = retObj.retVal;
 this.CleanUp();
 currRetValue=({err:false,retVal:psOutArgs});
 return currRetValue;
 }
 else {
 return
SiebelApp.PharmaCallValidatorsvc.superclass.InvokeMethod.call(this,
svcMthdName, psinpargs);
 }
 }
 PharmaCallValidatorsvc.prototype.CallValidate = function (psinpropset) {
 var currRetValue={err:false}, retObj;
```

```
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
 //Some Logic
 currRetValue=({err:false,retVal:psOutArgs});
 return currRetValue;
 };
 };
 return PharmaCallValidatorsvc;
 } ());
}
```

b. To make a call to a standalone service use the InvokeMethod method. Use the Client- Service Call method to customize the disconnected mobile client. This allows a service call to be made from the client, typically from a physical model.

The call from any other service file must be done as follows:

```
var service = SiebelApp.S_App.GetService("LS Pharma Validation Service");var
outputSet = service.Invoke("CallValidate", psPropertySet);
```

For example, the following code enables you to display the total number of products detailed in the tooltip. This would be the call from the physical model:

```
var service = SiebelApp.S_App.GetService("LS Pharma Validation Service");
var inPropSet = SiebelApp.S_App.NewPropertySet();
if (service) {
 retObj=currRetValue=service.InvokeMethod("CountPDMethod", inPropSet);
 var outPropSet = retObj.retVal;
}
```

In online mode, the call is to the standalone business service in a server, whereas in offline mode, this invokes the standalone offline business service code.

For example, the following code is for the Sample Offline service:

```
PharmaCallValidatorsvc.prototype.CanInvokeMethod = function (svcMthdName) {
 var currRetValue={err:false}, retObj;
 if (svcMthdName === " CountPDMethod") {
 currRetValue={ err: false, retVal: true };
 return currRetValue;
 }
 else {
 return
SiebelApp.PharmaCallValidatorsvc.superclass.CanInvokeMethod.call(this,
svcMthdName);
 }
 };
 PharmaCallValidatorsvc.prototype.InvokeMethod = function (svcMthdName,
Inputs) {
 var currRetValue={err:false}, retObj;
 var psOutArgs = CCFMiscUtil_CreatePropSet();
 if (svcMthdName === " CountPDMethod") {
 var BO = SiebelApp.S_App.GetBusObject("Pharma Professional Call -
Mobile");
 var PDBC = BO.GetBusComp("Pharma Call Products Detailed");
 PDBC.SetSearchExpr( "[Activity Id] = '" + Inputs.GetProperty("Id") +
"'");
 retObj=currRetValue=PDBC.ExecuteQuery();
 retObj=currRetValue=PDBC.FirstRecord();
 var result = PDBC.CountRecords();
 Outputs.SetProperty("OutputText",result);
 }
```

For more information about the methods that this step uses, see the following topics:

- *Invoke Method for Business Services*
- *InvokeMethod Method for Applets*

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 Mobile Guide: Disconnected* .

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 that describes configuring application settings in *Siebel Mobile Guide: Disconnected* .

Configuring Error Messages for Disconnected Clients

This topic describes how to configure Siebel Open UI to use the SetErrorMsg 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*.
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 ()
    var currRetValue={err:false}, retObj;
    retObj=currRetValue=this.Called();
   ```

   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){
 currRetValue=retObj;
 }
 else{
 //Positive case
 currRetValue={err:false,retVal:false};
 }
 });
 return currRetValue;
}
```

This code determines whether or not the Called method returns an error message. If it:

- Returns an error message, then this code calls the return value to some error.
- Does not return an error message, then the following code sets the err return value to null:

  ```
  currRetValue={err:false,retVal :false};
  ```

4. Add the following code to the code that you located in Step 3:

```
BusComp.prototype.Called = function (){
 var currRetValue={err:false}, retObj;
 var errParamArray = [];
 errParamArray.push(value1, valueN);
SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("messageKey", errParamArray);
currRetValue={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*.
  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. This value contains the error code:

  ```
  currRetValue = {err:"AppropriateErrorCode",retVal:false};

  return currRetValue;
  ```

  The following code is the completed code that this example uses:

  ```
  BusComp.prototype.Caller = function ()
   var currRetValue={err:false}, retObj;
   retObj=currRetValue=this.Called();
   //Check for any errors
   if(retObj.err){
   currRetValue=(retObj);
   }
   else{
   //Positive case
   currRetValue={err:false,retVal :false};
   }
  ```

```
 return currRetValue;
}
BusComp.prototype.Called = function (){
 var currRetValue={err:false}, retObj;
 var errParamArray = [];
 errParamArray.push(fieldName);
 SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("ErrorCode", errParamArray);
 currRetValue={err:"AppropriateErrorCode",retVal:false};
 return currRetValue;
}
```

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 an local folder. For example, swemessages_enu.js. For more information about the language_code, see *Languages That Siebel Open UI Supports*.
- 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."
```

About Siebel Mobile Application Logging

Users can enable logging for Siebel Mobile applications on their devices. For information about Siebel Mobile Application logging, see *Siebel Mobile Guide: Disconnected* .

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 Mobile Guide: Disconnected* .

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*. For more information about the methods that this example uses, see *Methods You Can Use to Customize Siebel Mobile Disconnected*.

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*.
2. Add the following code:

```
SiebelApp.pharmacallsvc = (function () {
 function pharmacallsvc(pm) {
 }
 SiebelJS.Extend(pharmacallsvc, SiebelApp.ServiceModel);
```

   This code adds the pharmacallsvc method to the pharmacallsvc business service.
3. Specify the logic for your method.
4. Add the following code immediately after the code you added in Step 3:

```
pharmacallsvc.prototype.InvokeMethod = function (psInputArgs) {
 var currRetValue={err:false}, retObj;
 var svcMthdName = "";
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
 svcMthdName = psInputArgs.GetProperty("MethodName").toString();
 if (svcMthdName === "Submit") {
 retObj=currRetValue=this.Submit();
 psOutArgs.SetProperty("Invoked", true);
 currRetValue={err: false, retVal: psOutArgs};
 }
 return currRetValue;
};
```

   This code configures Siebel Open UI to run InvokeMethod on the business service if the svcMthdName variable that you defined in Step 3 contains a value of Submit.
5. Define the method that includes your customization logic. You add the following code immediately after the code you added in Step 4:

```
pharmacallsvc.prototype.Submit = function () {
 var currRetValue={err:false}, retObj;
 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").GetPickAppl
et();
```

```
 retObj=currRetValue=pBusComp.SetFieldValue(strStatusField, "submit", true);
 retObj=currRetValue=pBusComp.WriteRecord();
 return currRetValue;
}
```

This code defines the Submit method. It sets the value for the Status field to Submitted. It uses the following methods:

- *BusComp Method for Applets*
- *SetFieldValue Method*
- *WriteRecord Method*
- *GetActiveView Method*

6. 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.

Configuring Interactive Detailing in the Siebel Open UI Application for Siebel Pharma

Configuring interactive detailing involves configuring the Detail button to appear on an applet in the application. By default, the Detail button appears only for Calls in the Siebel Open UI application for Siebel Pharma. Selecting the Detail button starts the eDetailer player which is used to deliver personalized content to customers, to demonstrate information about products to customers, and to obtain feedback from customers about product presentations and personalized content delivered. For more information about using the eDetailer player in the Siebel Open UI application for Siebel Pharma, see *Siebel Mobile Connector Guide* .

Configuring the Detail Link - Scenario 1: Using New Data Map Object to Capture Customer Feedback

The following procedure shows you how to configure the Detail link for Contacts in the Siebel Open UI application for Siebel Pharma, but you follow the same procedure if configuring the Detail link for any other applet in the application. In the following procedure, you configure a new data map object (EdetailingContact) to create the Activity and Response record to capture customer feedback.

To configure the Detail link for Contacts in the Siebel Open UI application for Siebel Pharma

1. Create a new Detail button control and drilldown in the Contact Form 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 Contact Form Applet.
d. Create a new Detail button control:

   - In the Object Explorer, expand the Contact Form Applet, and then Control.
   - In the Controls list, create a new button control using values from the following table.

Property	Value
Name	EdetailerButton
Caption	Detail
Method Invoked	ShowEdetaillerPreviewView This method handles the related view navigation and data for the Detail link (eDetailer player). ShowEdetailerPreviewView is a new LS PCD Service for delivering personalized content in the Life Sciences industry. Note that if Siebel Tools does not display the Method Invoked in the list, then type it in manually.

e. Define user properties for the Detail button:

   - In the Object Explorer, expand the Controls tree, and then click Business Component User Prop.
   - If you are invoking the business service method Named Method, then the user property value for Named Method is as follows:

User Property Name	Value
Named Method 1	"ShowEdetailerPreviewView", "INVOKESVC", "Contact", "LS PCD Service", "ShowEdetailerPreviewView", "DrilldownName", "Edetailer Drilldown", "EdetailerDatamapObj", "EdetailingContact", "CreateBookmark", "true", "'ObjectId'", "[Id]"

   - Create input arguments for Named Method with the values shown in the following table.

Property Name	Value	Purpose
DrilldownName	Edetailer Drilldown	Navigates to the eDetailer player view.

Property Name	Value	Purpose
EdetailerDatamapObj	EdetailingContact	Triggers the creation of activities, and the feedback capture page when finished showing the presentation.
CreateBookmark	TRUE	Navigates back to the originating view (for example, Contact) when done showing the presentation.
ObjectId	Row Id of current record	Used to log the response captured to the appropriate contact or account call.

f. Add a new drilldown object for the Detail button control:

- In the Object Explorer, expand the Contact Form Applet, and then Drilldown Object.
- In the Drilldown Objects list, add a new drilldown object with the values shown in the following table.

Property	Value
Name	Edetailer Drilldown
Hyperlink Field	Last Name
View	eDetailer Message Plan Preview View
Source Field	None
Business Component	LS Admin Messagign Plans BC

To show only the messaging plans that are related to a particular object (that is, remove the object for example "Product"), then add a new drilldown object with the values shown in the following table.

Property	Value
Name	Edetailer Drilldown
Hyperlink Field	Name

Property	Value
View	eDetailer Message Plan Preview View
Source Field	Id
Business Component	LS Admin Messaging Plans BC
Destination Field	Product Id

2. Add the Contact business component to the Admin Messaging Plan business object.
   a. In the Object Explorer, expand the Business Object tree, and then click Business Object Component.
   b. In the Business Object Component list, create new records with the values shown in the following table.

Business Object Component	Value
Bus Comp	Link
Contact	None

3. Configure a new data map object (EdetailingContact) to create the Activity and Response record:

> **Note:** Data Maps can be Workspace enabled in your development environment. If they have been Workspace enabled in your development environment and you are working in that environment, then you can only modify them in an editable Workspace. You do not need an editable Workspace to create and edit Data Maps in your Production environment.

   a. Log in to the Siebel business application.
   b. Navigate to the Administration - Application screen, then the Data Map Administration view.
   c. Click New and create a new data map object with the values shown in the following table:

Data Map Object Name	Source Business Object	Destination Business Object
EdetailingContact	Admin Messaging Plan	Action

Data Map Object Name	Source Business Object	Destination Business Object

d. For the EdetailingContact data map object, click New in the Data Map Component applet and add the following components:

Name	Source Business Component	Destination Business Component	Parent	Advanced Options
Contact Act	Contact	Action	None	Source Search Specification = [Id] = GetProfileAttr ('Edetailer Object Id')
ResponseLog	eDetailer Feedback Capture VBC	LS PCD Presentation Details BC	Contact Act	None

e. For the Contact Act data map component, click new in the Data Map Field applet and add the following fields:

Source Type	Source	Destination Type	Destination
Field	Id	Field	Primary Contact Id

f. For the ResponseLog data map component, click new in the Data Map Field applet and add the following fields:

Source Type	Source	Destination Type	Destination
Field	EndTime	Field	Message End Time
Expression	GetProfileAttr("Edetailer Object Id")	Field	Contact Id
Field	ItemName	Field	Message
Field	MpiId	Field	Message Id
Field	ParentMPId	Field	Message Plan Id
Field	ParentMPName	Field	Message Plan

Source Type	Source	Destination Type	Destination
Field	StartTime	Field	Message Start Time

Configuring the Detail Link - Scenario 2: Using New Business to Capture Customer Feedback

The following procedure shows you how to configure the Detail link in the Siebel Open UI application for Siebel Pharma specifically. To configure the Detail link in a different Siebel Open UI application (for example, in the Siebel Open UI application for Siebel Service), follow the procedure shown in *Configuring the Detail Link - Scenario 1: Using New Data Map Object to Capture Customer Feedback*. In the following procedure, you configure new business component user properties (rather than a new data map object) to capture customer feedback.

To configure the Detail link for Contacts in the Siebel Open UI application for Siebel Pharma

1. Create a new Detail button control and drilldown in the Contact Form 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 Contact Form Applet.
   d. Create a new Detail button control:
      - In the Object Explorer, expand the Contact Form Applet, and then Control.
      - In the Controls list, create a new button control using values from the following table.

Property	Value
Name	EdetailerButton
Caption	Detail
Method Invoked	ShowEdetaillerPreviewView This method handles the related view navigation and data for the Detail link (eDetailer player). ShowEdetailerPreviewView is a new LS PCD Service for delivering personalized content in the Life Sciences industry. Note that if Siebel Tools does not display the Method Invoked in the list, then type it in manually.

Property	Value

e. Define user properties for the Detail button:

- In the Object Explorer, expand the Controls tree, and then click Business Component User Prop.
- If you are invoking the business service method *Named Method*, then the user property value for Named Method is as follows:

User Property Name	Value
Named Method 1	"ShowEdetailerPreviewView", "INVOKESVC", "Pharma Professional Call", "LS PCD Service", "ShowEdetailerPreviewView", "DrilldownName", "Edetailer Drilldown", "CreateBookmark", "true", "'ObjectId'", "[Id]"

- Create input arguments for Named Method with the values shown in the following table:

Property Name	Value	Purpose
DrilldownName	Edetailer Drilldown	Navigates to the eDetailer player view.
CreateBookmark	TRUE	Navigates back to the originating view (for example, Contact) when done showing the presentation.
ObjectId	Row Id of current record	Used to log the response captured to the appropriate contact or account call.

f. Add a new drilldown object for the Detail button control:

- In the Object Explorer, expand the Contact Form Applet, and then Drilldown Object.
- In the Drilldown Objects list, add a new drilldown object using values from the following table.

Property	Value
Name	Edetailer Drilldown
View	eDetailer Message Plan Preview View
Hyperlink Field	Last Name
Source Field	None

Property	Value
Business Component	LS Admin Messaging Plans BC

To show only the messaging plans that are related to a particular object, then add a new drilldown object with the values shown in the following table.

Property	Value
Name	Edetailer Drilldown
Hyperlink Field	Name
View	eDetailer Message Plan Preview View
Source Field	Id
Business Component	LS Admin Messaging Plans BC
Destination Field	Product Id

2. Add the Contact business component to the Admin Messaging Plan business object.
   a. In the Object Explorer, expand the Business Object tree, and then click Business Object Component.
   b. In the Business Object Component list, create new records with the values shown in the following table.

Business Object Component	Value
Bus Comp	Link
Contact	None

3. Configure the business component user properties with the values shown in the following table for the eDetailer Feedback Capture VBC business component:

Business Component User Property	Value
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 1	EndTime|Message End Time

Business Component User Property	Value
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 2	ItemName|Message
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 3	MpiId|Message Id
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 4	ParentMPId|Message Plan Id
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 5	ParentMPName|Message Plan
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 6	StartTime|Message Start Time
eDetailer Feedback Capture VBC|LS PCD Presentation Details BC FieldMap 7	Response|Respons
SourceBC	eDetailer Feedback Capture VBC
DestinationBC	LS PCD Presentation Details BC

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*
- *Allowing Users to Return Parts*
- *Allowing Users to Set the Activity Status*

For more information about:

- Work you do in this topic, see *Process of Customizing Siebel Open UI for Siebel Mobile Disconnected*
- Methods that these examples use, see *Methods You Can Use to Customize Siebel Mobile Disconnected*
- Functionality that these customizations modify, see the chapter that describes how to use the Siebel Mobile Disconnected Application for Siebel Service in *Siebel Mobile Guide: Disconnected*

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:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\offline
   ```

2. Copy the servicecommitpartconsumed.js file to the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\files\custom\
   ```

   For more information, see *Organizing Files That You Customize*.
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 (SiebelApp.commitpartconsumed) === "undefined") {
   SiebelJS.Namespace('SiebelApp.commitpartconsumed');
   ```

5. Add the following code immediately after the code that you located in Step 4:

   ```
   var inputArgs = {};
    var oconsts = SiebelApp.Offlineconstants;
    inputArgs[oconsts.get("DOUIREG_OBJ_NAME")]= "SHCE Service FS Activity Part Movements List Applet -
    Mobile";
    inputArgs[oconsts.get("DOUIREG_OBJ_TYPE")]= oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
    inputArgs[oconsts.get("DOUIREG_OBJ_MTHD")]= "CommitPartMvmtClient";
    inputArgs[oconsts.get("DOUIREG_SRVC_NAME")]= "commitpartconsumed";
    inputArgs[oconsts.get("DOUIREG_SRVC_MTDH")] = "CommitPartMvmtClient";
    inputArgs[oconsts.get("DOUIREG_EXT_TYPE")]= null;
    SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);
   ```

   This code registers the service. For more information, see *ServiceRegistry Method*.
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, psinpargs) {
 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, psinpargs);
 }
 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 pServiceInvBC;
 var cszCommitFlag;
 var pModel;
pModel = SiebelApp.S_App.Model;
var pServiceInvBO = pModel.GetBusObject("boName");
pServiceInvBC = pServiceInvBO.GetBusComp("bcName");
cszCommitFlag = pServiceInvBC.GetFieldValue("Commit Txn Flag");
if (cszCommitFlag === 'Y'){
 SiebelJS.Log('Consumed Part Is Already In Committed State');
 }
 else
 {
 // pServiceInvBC.ActivateField("Commit Txn Flag");
 //pServiceInvBC.UpdateRecord();
 pServiceInvBC.SetFieldValue("Commit Txn Flag", "Y", true);
 pServiceInvBC.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*, *SetFieldValue Method*, and *WriteRecord Method*.

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 4 through Step 10. 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:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\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.commitpartconsumed) === "undefined") {
    SiebelJS.Namespace('SiebelApp.commitpartconsumed');
    var inputArgs = {};
    var oconsts = SiebelApp.Offlineconstants;

    inputArgs[oconsts.get("DOUIREG_OBJ_NAME")]="SHCE Service FS Activity Part
   Movements List Applet - Mobile";

   inputArgs[oconsts.get("DOUIREG_OBJ_TYPE")]=oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
    inputArgs[oconsts.get("DOUIREG_OBJ_MTHD")]="CanInvokeMethod";
    inputArgs[oconsts.get("DOUIREG_SRVC_NAME")]="commitpartconsumed";
    inputArgs[oconsts.get("DOUIREG_SRVC_MTDH")]="CanInvokeMethod";
    inputArgs[oconsts.get("DOUIREG_EXT_TYPE")]=oconsts.get("DOUIREG_EXT_TYPEPRE");
    SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);

    inputArgs={};

    inputArgs[oconsts.get("DOUIREG_OBJ_NAME")]="SHCE Service FS Activity Part
   Movements List Applet - Mobile";

   inputArgs[oconsts.get("DOUIREG_OBJ_TYPE")]=oconsts.get("DOUIREG_OBJ_TYPEAPPLET");
    inputArgs[oconsts.get("DOUIREG_OBJ_MTHD")]="InvokeMethod";
    inputArgs[oconsts.get("DOUIREG_SRVC_NAME")]="commitpartconsumed";
    inputArgs[oconsts.get("DOUIREG_SRVC_MTDH")]="InvokeMethod";
    inputArgs[oconsts.get("DOUIREG_EXT_TYPE")]=oconsts.get("DOUIREG_EXT_TYPEPRE");

    SiebelApp.S_App.GetModel().ServiceRegistry(inputArgs);

    inputArgs={};
   ```

   For information about the methods that this code uses, see the following topics:

- *CanInvokeMethod Method*
- *ServiceRegistry Method*
- *InvokeMethod Method for Applets*

5. Add the following code. This code calls the CanInvokeMethod method:

```
SiebelApp.commitpartconsumed = (function () {
 function commitpartconsumed(pm) {
 }
 var commitObj = new commitpartconsumed();
 commitpartconsumed.prototype.CanInvokeMethod = function (psInputArgs) {
 var currRetValue={err:false}, retObj;
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
 var svcMthdName = "";
 svcMthdName = psInputArgs.GetProperty("MethodName").toString();
 if (svcMthdName === "CommitPartMvmtClient") {
 psOutArgs.SetProperty("Invoked", true);
 psOutArgs.SetProperty("RetVal", true);
 currRetValue={err:false,retVal:psOutArgs};
 }
 else if (svcMthdName === "OrderPartsRMA") {
 psOutArgs.SetProperty("Invoked", true);
 psOutArgs.SetProperty("RetVal", true);
 currRetValue={err:false,retVal:psOutArgs};
 }
 else{
 psOutArgs.SetProperty("Invoked", false);
 psOutArgs.SetProperty("RetVal", false);
 currRetValue={err:false,retVal:psOutArgs};
 }
 return currRetValue;
 };
```

6. Add the following code. This code calls the InvokeMethod method:

```
commitpartconsumed.prototype.InvokeMethod = function (psInputArgs) {
 var currRetValue={err:false}, retObj;
 var svcMthdName = "";
 var psOutArgs = SiebelApp.S_App.NewPropertySet();
 svcMthdName = psInputArgs.GetProperty("MethodName").toString();
 if (svcMthdName === "CommitPartMvmtClient") {
 retObj=currRetValue=this.CommitPartMvmtClient();
 psOutArgs.SetProperty("Invoked", true);
 currRetValue={err:false,retVal:psOutArgs};
 }
 else{
 psOutArgs.SetProperty("Invoked", false);
 currRetValue={err:false,retVal:psOutArgs};
 }
 if (svcMthdName === "OrderPartsRMA") {
 retObj=currRetValue=this.OrderPartsRMA();
 psOutArgs.SetProperty("Invoked", true);
 currRetValue={err:false,retVal:psOutArgs};
 }
 else{
 psOutArgs.SetProperty("Invoked", false);
 currRetValue={err:false,retVal:psOutArgs};
 }
 return currRetValue;
 };
```

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:

```
commitpartconsumed.prototype.createRMAOrder = function (orderType) {
 var currRetValue={err:false}, retObj;
 var sOrderId;
 var cszOrderId;
 var sAssetNum;
 var sPartNum;
 var sStatus;
 var sProductId;
 var sProductName;
 var sQuantity;
 var sActivityPartMvmtID;
 var pModel;
 var pFSActivityPartsMovementBC;
 var pActionBC;
 var sSR_Id;
 var pServiceRequestBC;
 var pOrderEntry_OrdersBC;
 var pOrderEntry_LineItemBC;
 var errParamArray = [];
 pModel = SiebelApp.S_App.Model;
 var pBusObj = pModel.GetBusObject("boName")
 pFSActivityPartsMovementBC=pBusObj.GetBusComp("bcName");
 sOrderId=retObj.retVal;
 if (utils.IsEmpty(sOrderId)){
 retObj=currRetValue=pFSActivityPartsMovementBC.GetFieldValue("");
 var oPsDR_Header:PropertySet = 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 lPS_values:PropertySet = SiebelApp.S_App.NewPropertySet();
 oPsDR_Header.SetProperty("Product Id","");
 oPsDR_Header.SetProperty("Used Quantity","");
 oPsDR_Header.SetProperty("Id","");
 oPsDR_Header.SetProperty("Asset Number","");
 oPsDR_Header.SetProperty("Part Number","");
 sPartNum=retObj.retVal;
 pActionBC=SiebelApp.S_App.GetActiveView().GetActiveApplet().BusComp().ParentBuscomp();
 retObj=currRetValue=pActionBC.GetFieldValue("Activity SR Id");
 sSR_Id=retObj.retVal;
 if(sSR_Id==""){
 //Activity has no associated SR... Hence the operation will be aborted
 SiebelApp.S_App.OfflineErrorObject.SetErrorMsg("IDS_ERR_FS_MISSING_SR", errParamArray);
 currRetValue={err: "IDS_ERR_FS_MISSING_SR", retVal:""};
 return currRetValue;
 }
 }
 return currRetValue;
```

```
}
```

For information about the methods that this code uses, see *GetFieldValue Method*.

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 = SiebelApp.S_App.Model;
 pServiceRequestBC = pModel.BusObj("Service Request").BusComp("Service Request");
 pOrderEntry_OrdersBC = SiebelApp.S_App.Model.GetBusObj("Service Request").BusComp("Order Entry -
 Orders");
 pOrderEntry_LineItemBC = pModel.BusObj("Service Request").BusComp("Order Entry - Line Items");
 //CREATE ORDER Header.
 retObj=currRetValue=pOrderEntry_OrdersBC.ExecuteQuery();
```

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:

```
retObj=currRetValue=pOrderEntry_OrdersBC.NewRecord(true);
sLocaleVal = SiebelApp.S_App.Model.GetLovNameVal(orderType, "FS_ORDER_TYPE");
retObj=currRetValue=pOrderEntry_OrdersBC.SetFieldValue("Order Type", sLocaleVal, true);
retObj=currRetValue=pOrderEntry_OrdersBC.WriteRecord();
retObj=currRetValue=pOrderEntry_OrdersBC.GetFieldValue("Id");
sOrderItemId=retObj.retVal;
retObj=currRetValue=pOrderEntry_OrdersBC.GetFieldValue("Id");
m_sOrderHeaderId=retObj.retVal;
retObj=currRetValue=pOrderEntry_LineItemBC.ExecuteQuery();
```

   For information about the methods that this code uses, see *SetFieldValue Method*, *WriteRecord Method*, *NewRecord Method*.

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:

```
retObj=currRetValue=pOrderEntry_LineItemBC.NewRecord(true);
retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Product Id",
retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Product", sProductName, true);
retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Quantity Requested",sQuantity, true);
if(!utils.IsEmpty(sAssetNum)){
```

```
 retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Asset Number", sAssetNum, true);
}
if(!utils.IsEmpty(sPartNum)){
 retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Part Number", sPartNum, true);
}
if(!utils.IsEmpty(sStatus)){
 retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Product Status Code",sStatus, true);
}
retObj=currRetValue=pOrderEntry_LineItemBC.GetFieldValue("Id");
sOrderItemId=retObj.retVal;
retObj=currRetValue=pOrderEntry_LineItemBC.SetFieldValue("Order Header Id", m_sOrderHeaderId, true)
retObj=currRetValue=pOrderEntry_LineItemBC.WriteRecord();
retObj=currRetValue=pFSActivityPartsMovementBC.SetFieldValue("Order Item Id",sOrderItemId, true);

retObj=currRetValue=pFSActivityPartsMovementBC.WriteRecord();
```

11. Save, and then close the servicecmtparts.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*.

To allow users to set the activity status

1. In Windows Explorer, navigate to the following folder:

   ```
   AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\offline
   ```

   For more information about the language_code, see *Languages That Siebel Open UI Supports*.
2. Use a JavaScript editor to open the serviceactstat.js file.
3. Locate the following code:

   ```
   serviceactstat.prototype.InvokeSetActStatus=function(psInpArgs,svcMthdName){
    var psOutArgs=SiebelApp.S_App.NewPropertySet();
    if(!psInpArgs){
    return (false);
    }
    if(psInpArgs.propArray.MethodName=="AcceptStatus")
    {psOutArgs=this.SetActivityStatus("Acknowledged");
    }
    else if(psInpArgs.propArray.MethodName=="Start"||psInpArgs.propArray.
    MethodName=="ArrivedStatus"){psOutArgs=this.SetActivityStatus("In
    Progress","ArrivedStatus");
    }
    else if(psInpArgs.propArray.MethodName=="DeclineStatus"){
    psOutArgs=this.SetActivityStatus("Declined");
    }
    else if(psInpArgs.propArray.MethodName=="EnrouteStatus"){
    psOutArgs=this.SetActivityStatus("In Progress");
    }
    else if(psInpArgs.propArray.MethodName=="SuspendStatus"){
    psOutArgs=this.SetActivityStatus("On Hold");
    }
    else if(psInpArgs.propArray.MethodName=="ResumeStatus"){
    psOutArgs=this.SetActivityStatus("In Progress");
    }
    else if(psInpArgs.propArray.MethodName=="End"||psInpArgs.propArray.
    MethodName=="FinishedStatus"){
    psOutArgs = this.SetActivityStatus("Done","FinishedStatus");
    }
   ```

4. Add the following code immediately after the code you located in Step 3:

   ```
   serviceactstat.prototype.SetActivityStatus=function (pStatus,pDateMethodInv){
    var currRetValue={err:false}, retObj;
    SiebelJS.Log('Service Method SetActivityStatus...');
    var strstatvalue;
    var pickName;
    var pickListDef;
    var pModel;
    var pBusComp;
    pModel= SiebelApp.S_App.GetModel();
    var pBusObj = pModel.GetBusObject("boName");
    pBusComp = pBusObj.GetBusComp("bcName");

   pickName=SiebelApp.S_App.GetActiveView().GetActiveApplet().GetControl("Status").Ge
   tPickApplet();
    pickListDef=pickListDef = pBusComp.GetPickListInfo(pickName);
    pModel=SiebelApp.S_App.Model;
    strstatvalue=pModel.GetLovNameVal("Acknowledged", pickListDef.LOVType);
    currRetValue=pBusComp.ActivateField("Status");
    currRetValue=pBusComp.SetFieldValue("Status",strstatvalue,true);
    currRetValue=pBusComp.ActivateField("Status");
    currRetValue=pBusComp.SetFieldValue("Status",strstatvalue,true);
   ```

```
 if(pDateMethodInv!="")//Todo - Refine this condition for uninitialized/defined or
remove this condition
 {
 var now=new Date();
 if(pDateMethodInv == "ArrivedStatus") {
 currRetValue=pBusComp.SetFieldValue("Started",now,true);
 currRetValue=pBusComp.SetFieldValue("Done","",true);
 }
 else if(pDateMethodInv=="FinishedStatus") {
 currRetValue=pBusComp.SetFieldValue("Done",now,true);
 currRetValue=pBusComp.SetFieldValue("Percent Complete","100%",true);
 }
 }
 currRetValue=pBusComp.WriteRecord();
 return currRetValue;
};
```

For information about the methods that this code uses, see the following:

- *SetFieldValue Method*
- *WriteRecord Method*
- *GetActiveView Method*

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*
- *Methods You Can Use in the Business Component Class*
- *Methods You Can Use in the Business Object Class*
- *Methods You Can Use in the Business Service Class*
- *Methods You Can Use in the Application Class*
- *Methods You Can Use in the Model Class*
- *Methods You Can Use in the Service Model Class*
- *Methods You Can Use in Offline Classes*
- *Other Methods You Can Use with Siebel Mobile Disconnected*

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*.

Methods You Can Use in the Applet Class

This topic describes methods that you can use that reside in the Applet class. The methods are described in the following subtopics.

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*.

BusObject Method for Applets

The BusComp method returns the business component that the applet 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*
- *Using Custom Siebel Business Services*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Return Parts*

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* and *Allowing Users to Commit Part Tracker Records*.

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. The methods are described in the following subtopics.

ActivateField Method

The ActivateField method activates a business component field. It returns nothing. It uses the following syntax:

```
this.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 file 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 currRetValue={err:false}, retObj;
var model= SiebelApp.S_App.GetModel();
var boContact = model.GetBusObject("Contact");
var bcContact = boContact.GetBusComp("Contact");
bcContact.ClearToQuery();
currRetValue=bcContact.ActivateField("Login Name");
var sLoginName = "MYNAME";
bcContact.SetSearchSpec("Login Name", sLoginName);
retObj=currRetValue=bcContact.ExecuteQuery();
if (!retObj.err) {
 model.ReleaseBO(boContact);
}
```

ActivateMultipleFields Method

The ActivateMultipleFields method activates more than one field. It returns nothing. It uses the following syntax:

```
BusComp.ActivateMultipleFields(SiebelPropertySet);
```

where:

- `SiebelPropertySet` 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 = SiebelApp.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 currRetValue={err:false}, retObj;
var model= SiebelApp.S_App.GetModel();
var ContactBC = model.GetBusObject("Contact");
var ContactBC = boContact.GetBusComp("Contact");
if (ContactBC)
 {
 var fieldsPS = SiebelApp.S_App.NewPropertySet();
 var valuesPS = SiebelApp.S_App.NewPropertySet();
 fieldsPS. SetProperty("Last Name", "");
 fieldsPS.SetProperty("First Name", "");
 ContactBC.ActivateMultipleFields(fieldsPS);
 ContactBC .ClearToQuery();
 currRetValue=ContactBC.ExecuteQuery();
 if (!retObj.err) {
 retObj=currRetValue=ContactBC.FirstRecord();
 if (!retObj.err) {
 ContactBC .GetMultipleFieldValues(fieldsPS, valuesPS);
 var slName = valuesPS.GetProperty("Last Name");
 var sfName = valuesPS.GetProperty("First Name");
 }
 }
}
return currRetValue;
```

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:

```
SiebelApp.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*.

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*.
- 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= SiebelApp.S_App.GetModel();
var oEmpBusObj= model.GetBusObject("Employee");
var oEmpBusComp = oEmpBusObj.GetBusComp("Employee ");
var sLoginName;
oEmpBusComp.ClearToQuery();
oEmpBusComp.SetSearchSpec("Login Name", sLoginName);
oEmpBusComp.ExecuteQuery();
```

For another example usage of the ClearToQuery method, see *CountRecords Method*.

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 currRetValue={err:false}, retObj;
var model= SiebelApp.S_App.GetModel();
var bo = model.GetBusObject("Opportunity ");
var bc = bo.GetBusComp("Opportunity");
if (bc)
 {
 bc .ClearToQuery();
 bc .SetSearchSpec ("Name", "A");
```

```
retObj=currRetValue=bc.ExecuteQuery();
if (!retObj.err) {
var count = bc .CountRecords();
currRetValue={err:false,retVal:count};
}
}
return currRetValue;
```

For more information, see *ClearToQuery Method*.

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:

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*.

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:

```
SiebelApp.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*.

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 currRetValue={err:false}, retObj;
var Model =SiebelApp.S_App.GetModel()
varaccount BO = Model.GetBusObj("Account");
var accountBC = accountBO.GetBusComp("Account");
var contactBC = accountBO.GetBusComp("Contact");
accountBC.SetSearchSpec("Id",[AccountId]);
currRetValue=accountBC.ExecuteQuery ();
currRetValue=accountBC.FirstRecord();// positions on the account record
currRetValue=contactBC.ExecuteQuery ();
currRetValue=contactBC.FirstRecord();
var assocBC = contactBC.GetAssocBusComp();
assocBC.SetSearchSpec("Id",[ContactID]);
currRetValue=assocBC. ExecuteQuery ();
currRetValue=assocBC.FirstRecord();// positions on the contactbc
```

```
currRetValue=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.GetFieldValue("field_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.GetFieldValue "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.GetFieldValue("Account Name",recordObject)
```

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* and *SetErrorMsg Method*.

For more examples that use the GetFieldValue method, see the following topics:

- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Commit Part Tracker Records*
- *Allowing Users to Return Parts*

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
- primeIdFieldName
- 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.GetLastErrCode()
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().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 Components

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:

```
BusComp.GetLastErrText()
```

For example:

```
ActiveBusObject().GetLastErrText();
```

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(fieldNamesPropSet, fieldValuesPropSet)
```

where:

- `fieldNamesPropSet` is a property set that identifies a collection of fields.
- `fieldValuesPropSet` is a property set that includes values for the fields that the fieldNamesPropSet 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 fieldNamesPropSet 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 lPS_values = SiebelApp.S_App.NewPropertySet();
oPsDR_Header.SetProperty("Last Name","");
oPsDR_Header.SetProperty("First Name","");
oPsDR_Header.SetProperty("Middle Name","");
var currRetValue={err:false}, retObj;
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*");
currRetValue=ExecuteQuery();
currRetValue=FirstRecord();
// Use a different property set for the values. If you use the same one
// for arguments you get no values back.
currRetValue=GetMultipleFieldValues(oPsDR_Header, lPS_values);
// Get the value from the output property set.
SiebelJS.Log("FullName is " +lPS_values.GetProperty("First Name") +
lPS_values.GetProperty("Middle Name")+ lPS_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:

```
retObj=currRetValue=this.GetFieldValue("City")
if(retObj.retVal === "San Mateo")
{
 var oBCPick = this.GetPicklistBusComp("State");
 oBCPick.SetSearchSpec("Value", "CA");
 oretObj=currRetValue=oBCPick.ExecuteQuery(ForwardOnly);
 retObj=currRetValue=oBCPick.FirstRecord();
 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*.

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(FieldName);
```

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().GetSearchSpec (FieldName);
```

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:

```
SiebelApp.S_App.FindApplet(appletName).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. GetViewMode()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().GetViewMode();
```

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*.

Name Method for Business Components

The Name method returns the name of a business component. It uses the following syntax:

```
SiebelApp.S_App.FindApplet(appletName).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*. 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:

```
currRetValue=buscomp.InvokeMethod("RefreshBusComp");
retObj=currRetValue;
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:

```
currRetValue=buscomp.InvokeMethod("RefreshRecord");
retObj=currRetValue;
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.

```
SetFieldValue(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*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Commit Part Tracker Records*
- *Allowing Users to Return Parts*
- *Allowing Users to Set the Activity Status*

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 currRetValue={err:false}, retObj;
varmodel = 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);
currRetValue=bc.NewRecord();
currRetValue=bc.SetMultipleFieldValues(ps);
ps = null;
currRetValue=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(inMode);
```

where:

- `inMode` 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:

```
SiebelApp.S_App.FindApplet(appletName).BusComp().SetViewMode(inMode);
```

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:

```
SiebelApp.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:

```
this.UpdateRecord();
```

where:

- `this` 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 `this` specifies:

```
this. UpdateRecord(false)
```

The UpdateRecord method can run in a Siebel Mobile Disconnected client.

For more information, see *CanUpdate Method*.

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.writerecord(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*
- *Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Commit Part Tracker Records*

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 currRetValue={err:false}, retObj;
var model= SiebelApp.S_App.GetModel();
var bo = model.GetBusObject("Opportunity ");
var bc = bo.GetBusComp("Opportunity");
varstrDEANumber = 9089;
var strDEAExpDate = 02/12/2013;
currRetValue=bc.SetFieldValue("DEA#", strDEANumber);
retObj=currRetValue;
if (!retObj.err) {
 currRetValue=bc.SetFieldValue("DEA Expiry Date", strDEAExpDate);
 retObj=currRetValue;
 if (!retObj.err) {
 currRetValue=bc.SetFieldValue("DEA Expiry Date", strDEAExpDate);
 retObj=currRetValue;
 if (!retObj.err) {
 currRetValue=bc.WriteRecord();
 }
 }
}
```

Methods You Can Use in the Business Object Class

This topic describes methods that you can use that reside in the Business Object class. The methods are described in the following subtopics.

GetBusComp Method for Business Objects

The GetBusComp method returns the business component instance that a business object references. It uses the following syntax:

```
SiebelApp.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:

```
SiebelApp.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*. 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. The methods are described in the following subtopics.

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*.
- `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*.

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*. 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 DOUIREG_OBJ_NAME property with a value of Pharma Call Entry Mobile:

```
inputObj [oconsts.get("DOUIREG_OBJ_NAME")] = "Pharma Call Entry Mobile";
```

The following code specifies the property name:

```
oconsts.get("DOUIREG_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*.

Properties You Must Include to Register Custom Business Services

The following table 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.

Properties	Value
DOUIREG_OBJ_NAME	The name of a custom business service. For example: `LS Pharma Validation Service`
DOUIREG_SRVC_NAME	The name of the JavaScript class that the custom business service references. For example: `PharmaCallValidatorsvc`

The following table describes the properties you must include in the inputObj argument of the ServiceRegistery method when Siebel Open UI registers a custom business service that references a predefined applet or a predefined business component.

Property	Value
DOUIREG_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.
DOUIREG_OBJ_MTHD	Name of the predefined business service method that you must customize. For example, WriteRecord.
DOUIREG_SRVC_NAME	The name of the JavaScript class that the Class property of the business service method references. For example: `pharmacallsvc`
DOUIREG_SRVC_MTDH	Name of the business service method that you customized. For example, WriteRecord.
DOUIREG_EXT_TYPE	You can use one of the following values: • **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. The methods are described in the following subtopics.

ActiveBusObject Method

The ActiveBusObject method returns the business object that the active view references. It uses the following syntax:

```
Application. ActiveBusObject()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.ActiveBusObject();
```

ActiveViewName Method

The ActiveViewName method returns the name of the active view. It uses the following syntax:

```
Application. ActiveViewName()
```

It includes no arguments.

For example:

```
SiebelApp.S_App. ActiveViewName();
```

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:

```
Application. CurrencyCode()
```

It includes no arguments.

For example:

```
SiebelApp.S_App. CurrencyCode();
```

FindApplet Method

The FindApplet method returns the active applet. It uses the following syntax:

```
Application. FindApplet(appletName)
```

where:

- `appletName` is a string that contains the name of the active applet.

For example, if the Contact List Applet is the current applet, then the appletName variable in the following code returns the name of this applet as a string:

```
SiebelApp.S_App.FindApplet(appletName);
```

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:

```
Application. GetBusObject(business_object_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:

```
SiebelApp.S_App. GetBusObject(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:

```
SiebelApp.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*.

The following example creates a business service instance of the LS Pharma Validation Service business service:

```
var service = SiebelApp.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:

```
SiebelApp.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:

```
SiebelApp.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:

```
SiebelApp.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:

```
SiebelApp.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:

```
Application. PositionId()
```

It includes no arguments.

For example:

```
SiebelApp.S_App. PositionId();
```

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:

```
Application. PositionName()
```

It includes no arguments.

For example:

```
SiebelApp.S_App. PositionName();
```

Methods You Can Use in the Model Class

This topic describes methods that you can use that reside in the Model class. The methods are described in the following subtopics.

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 loginid = SiebelApp.S_App.Model.GetLoginId();
```

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:

```
SiebelApp.S_App.Model.ReleaseBO(objBO);
```

where:

- `objBo` 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 a method you can use that resides in the offline classes. The command is described in the following section.

This method resides in the OfflineErrorObject class.

SetErrorMsg Method

The SetErrorMsg method defines an error message for a business service that you customize. It returns nothing. It uses the following Syntax:

```
SiebelApp.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*. 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*.

Other Methods You Can Use with Siebel Mobile Disconnected

This topic describes other methods that you can use with Siebel Mobile Disconnected. The methods are described in the following subtopics.

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:

```
SiebelApp.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 = SiebelApp.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:

```
SiebelApp.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:

```
SiebelApp.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:

```
SiebelApp.S_App.Model.GetLovValName(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:

```
SiebelApp.S_App.Model.GetLovValName("Call","TODO_TYPE")
```

The GetLovValName method resides in the model.js file. You cannot configure Siebel Open UI to override this method.

9 Siebel Web Component Framework

Siebel Web Component Framework

This chapter describes an overview of Web Component Framework. It includes the following topics:

- *Overview of the Siebel Web Component Framework*
- *Architecture of the Siebel Web Component Framework*
- *Execution Flow with Oracle JET Components*
- *Sample JSON Structure for UI Definition*
- *Customizing Using Oracle JET Components in the Web Component Framework*
- *Supported Controls in Siebel Web Component Framework*

Overview of the Siebel Web Component Framework

The Siebel Web Component Framework supports seamless integration with Oracle JET components and helps render a modern user interface through Redwood UX. By using this framework, you can enhance the Siebel user experience through reusable, consistent, and responsive user interface elements while continuing to use the **Siebel Open UI framework**.

The Siebel Web Component Framework exposes Oracle JET component functionality within the Siebel application context and enables seamless integration with Siebel application data. The framework binds Siebel metadata and data to the corresponding properties of Oracle JET components so that information can be displayed within the Siebel application context.

The framework also supports Oracle JET component methods and events, which allow components to respond to user interaction and changes in the underlying Siebel data. In addition, the framework supports client-side configuration of the user interface layout and HTML structure used to render Siebel objects, such as applets and views, by using Oracle JET components and Redwood UX design patterns.

This framework is designed to coexist with Classic Open UI. Existing applications that use Presentation Models and Physical Renderers continue to function without modification. Customers can selectively adopt the Siebel Web Component Framework for new views, dashboards, and other user experiences that require modern interaction patterns and visualization.

For an overview of how Siebel Open UI uses presentation models and physical renderers, see *How Siebel CRM Renders Div Containers on Siebel Servers*.

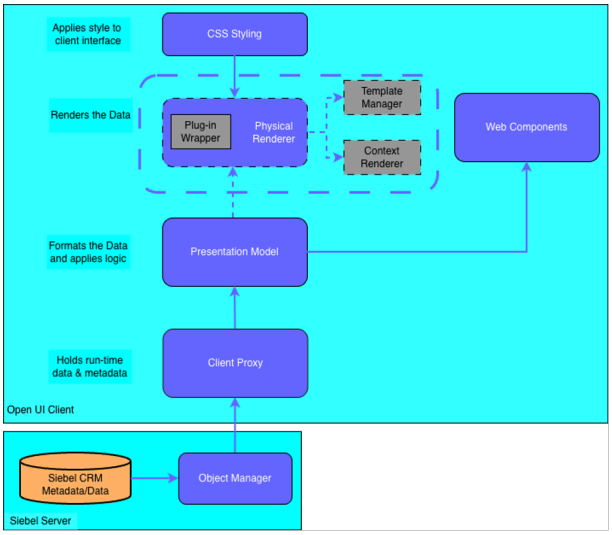
Architecture of the Siebel Web Component Framework

The Web Component Framework extends the Siebel Open UI architecture by introducing a client-side rendering model based on Web Components. In this model, the Presentation Model acts as the view model and exposes Siebel metadata, data, and behavior.

User interface composition is defined declaratively through JSON specifications that are provided through a Presentation Model property. Oracle JET Web Components consume this configuration to render the user interface and manage user interaction. When this framework is used, a Physical Renderer is not required for rendering the user interface.

Siebel Open UI supports user interface customization through Presentation Model and Physical Renderer objects that are configured in the manifest for views or applets. The Presentation Model provides a unified interface to Siebel metadata and data, and it supports customization through properties, methods, and event handling. The Physical Renderer supports user interface rendering and integration with third-party plug-ins when required.

The following figure illustrates the Siebel Web Component Framework architecture.



In the figure, a Web Components layer is introduced in parallel with the Physical Renderer (PR). The dotted lines indicate that the Physical Renderer is not required when the framework is used for rendering, but it remains part of the overall Siebel Open UI architecture.

With this architecture, the framework supports seamless integration of Web Components, primarily Oracle JET-based Web Components. The Presentation Model provides access to the data and metadata for the related object, such as an applet, and manages user interaction through properties, methods, and event handlers. The framework also provides a binding interface in the Presentation Model that directly binds Presentation Model properties, methods, and event handlers to Web Component properties, methods, and events.

Execution Flow with Oracle JET Components

The framework supports Oracle JET components and related Redwood page templates only when the active theme is Redwood.

The standard lifecycle of the Presentation Model (PM) and Physical Renderer (PR) through manifest configuration still applies. The following steps describe how the framework works:

1. When Siebel Open UI runs with the Redwood theme, the framework initializes Oracle JET components and the related Redwood page template support during application load.
2. For a Siebel applet or view, customization typically includes implementing a Presentation Model JavaScript file and configuring it for the corresponding object in the Manifest Administration view. In this scenario, you must add an expression so that the customization is applied only when the Redwood theme is active.
3. Implementing a Physical Renderer (PR) is not required when the Presentation Model (PM) interfaces with Oracle JET components, because the Web Components render the user interface by using the metadata and data that the PM provides. The framework reads the configured Presentation Model (PM) and Physical Renderer (PR) information from Manifest Administration.
4. The custom Presentation Model (PM) must extend the appropriate base class that the framework provides for the relevant use case.
5. The JSON configuration that is defined through the PM property SEBL_COMPONENT_CONFIG specifies the Oracle JET component configuration and the additional binding interface that maps metadata and data from Siebel objects, such as views, applets, and applet controls.
6. You can create additional properties and methods in the custom PM when they are required to support a specific use case or interaction from Oracle JET components.
7. The custom PM must be configured in the Manifest Administration view with the expression Redwood Theme for the corresponding object.
8. When the user navigates to the corresponding object, such as a view or applet, the framework processes the custom PM, supplies metadata and data according to the API specification, and lets the Oracle JET components render the user interface.
9. The framework provides a library of base PMs for various objects (List, Form, Chart, and Tree) and logical components for UI controls, such as buttons and text controls, that encapsulate standard Siebel functionality, including metadata and data consumption, to support reuse.

Sample JSON Structure for UI Definition

The SEBL_COMPONENT_CONFIG property in the Presentation Model (PM) defines the client-side structure that the Siebel Web Component Framework uses to render a user interface by using Oracle JET components. This JSON specification identifies the component to render, the attributes and bindings to apply, and the child content that must be composed within the layout.

Use this JSON specification to define the structure of a Siebel view or applet declaratively, without requiring user interface rendering logic to be implemented in a Physical Renderer. The framework interprets the JSON definition at run time, resolves the configured bindings to Siebel metadata and data, and then renders the corresponding Oracle JET components.

The following sections describe the properties that you can use in the SEBL_COMPONENT_CONFIG JSON specification.

A sample JSON specification for the Presentation Model property SEBL_COMPONENT_CONFIG appears as follows:

```
{
 "container": "#optional_container_selector",
 "imports": [],
 "variables": {},
 "componentDef": {},
 "component": "",
 "attributes": {},
```

```
 "children": []
}
```

- **container**

  Use the optional container property to specify the DOM selector where the framework must mount the rendered content. If you do not specify this property, then the framework uses the default run-time container behavior for the corresponding object.

  Definition of container:

  ```
  {
   "container": "#optional_container_selector"
  }
  ```

- **imports**

  Use the imports property to specify the modules that must be loaded before the framework renders the user interface. These modules typically include Oracle JET components and supporting layout modules that are required by the JSON configuration.

  Definition of imports:

  ```
  {
   "imports": [
   "oj-sp/foldout-layout/loader",
   "oj-sp/foldout-panel-summarizing/loader",
   "oj-sp/empty-state/loader",
   "oj-sp/horizontal-overview/loader",
   "oj-c/input-text",
   "ojs/ojformlayout"
   ]
  }
  ```

- **variables**

  Use the variables property to define key-value pairs that can be referenced anywhere within the configuration by using the $var: syntax.

  Definition of variables:

  ```
  {
   "variables": {
   "Applet": "Customer 360 Dashboard Account Form Applet",
   "itemTitle": "Name",
   "StreetAddress": "Street Address",
   "City": "City"
   }
  }
  ```

- **componentDef**

  Use the componentDef property to define reusable component configurations within the JSON specification. This property helps you centralize repeated user interface fragments and reuse them across the same configuration instead of redefining the same structure multiple times. Each entry in componentDef defines a reusable component name, the parameters that it accepts, and the configuration that the framework must render when that component is referenced.

  You can reference a reusable component that is defined in componentDef by using the $fn:component(...) syntax. Parameter values that are passed to the reusable component can be accessed within the component definition by using the $fn:param(...) syntax.

The config object in componentDef follows the same JSON semantics as the main configuration object. It can contain keys such as variables, component, children, attributes, and other supported configuration properties.

Use $fn:component to reference a reusable component that is defined in componentDef. When the framework evaluates $fn:component(ComponentName), it resolves the corresponding reusable component definition and renders the configuration that is associated with that component.

Definition of componentDef:

```
{
 "componentDef": {
 "ReusableComponent": {
 "params": ["data"],
 "config": {
 "component": "div",
 "children": [
 "$fn:param(data)"
 ]
 }
 }
 }
}
```

- **component**

  Use the component property to specify the HTML element, Oracle JET component, applet reference, or registered component that the framework must render.

  As an HTML element or Oracle JET component:

  ```
  "component": "oj-sp-foldout-layout"
  ```

  As a reference to an applet:

  ```
  "component": "$Applet:(AppletName)"
  ```

  This syntax indicates that the framework must render the referenced applet at that location in the DOM.

  As a reference to a component that is registered in the Model-View Factory:

  ```
  "component": "$fn:component(ComponentName)"
  ```

  This syntax indicates that the framework must render a special component that is registered in the Model-View Factory.

  > **Note:** Older configurations might use $WebComp:(ComponentName), such as $WebComp:(TextModelView), to reference a registered component. This syntax is retained for backward compatibility, but for new configurations use $fn:component(ComponentName).

- **attributes**

  Use the attributes property to define the properties, HTML attributes, and configuration values that the framework must apply to the component that is being rendered. Attribute values can be specified as static values or by using supported binding expressions, depending on the configuration requirement.

  Definition of attributes:

  ```
  {
   "attributes": {
   "class": "oj-flex",
  ```

```
 "label-edge": "start"
 }
}
```

- **children**

  Use the children property to define the child content that the framework must render within the parent component. The children property can contain static text, nested component definitions, or supported dynamic expressions. This property allows you to build the user interface hierarchically through nested JSON structures.

  Definition of children:

```
{
 "children": [
 "Customer Details",
 {
 "component": "div",
 "attributes": {
 "class": "oj-flex-item"
 },
 "children": [
 "Account Name"
 ]
 }
 ]
}
```

Points on Children Definition

The children array can directly take dynamic values, as described earlier, by using both long-form and shorthand syntax.

For rendering content such as the following example, notice that AppletControl-DisplayName becomes the text content in the div tag and AppletControl-Value becomes the text content in the b tag.

```
<div>
 AppletControl-DisplayName
 <b>
 AppletControl-Value
 </b>
</div>
```

The corresponding JSON configuration appears as follows:

```
{
 "children": [
 "$fn:value($, #1, DisplayName)",
 {
 "component": "b",
 "children": [
 "$fn:value($, #1, Value)"
 ]
 }
 ]
}
```

Dynamic Data Binding with Siebel Open UI

The Siebel Web Component Framework supports dynamic data binding so that JSON configuration can reference Siebel metadata, control values, Presentation Model properties, and derived values at run time. This binding model allows the framework to render content declaratively while continuing to use the Siebel Open UI Presentation Model as the source of data and behavior.

Dynamic binding can be specified by using shorthand syntax for common expressions or long-form syntax for advanced composition. These bindings can be used in properties such as children, attributes, and other configuration nodes where run-time values must be resolved.

Note: The AppletName (AppletPM) or ViewPM can almost always be referenced by using the "$" symbol, which indicates the Presentation Model for which the configuration is being written. This is useful when you do not want to repeat the current applet name and instead want to directly reference the current applet context. The AppletName and ControlName can also often be referenced by using the applet and control index, such as "#1" and "#2". This indicates that the reference uses the View Web Template Item or Applet Web Template Item position rather than the object name.

- **Shorthand Syntax: Data Binding with Siebel OpenUI**

  Use shorthand syntax when the binding expression can be represented directly without using a long-form object definition. Shorthand syntax is useful for common binding patterns and keeps the JSON configuration concise and readable.

  - **literals**

    Literals are static values that do not use an Open UI binding expression. A literal can be a string, array, object, Boolean value, or number.

    If the top-level object is a literal, then the entire object hierarchy is treated as a literal.
  - **$fn:value**

    Define binding to get data from an Applet Control. This can be the control's DisplayName, Value, or CanInvoke property.

    Possible syntax:

    - $fn:value(AppletName, ControlName, Value)
    - $fn:value(AppletName, ControlName, DisplayName)
    - $fn:value(AppletName, ControlName, CanInvoke)
    - $fn:value($, ControlName, Value)
    - $fn:value($, #2, DisplayName)
  - **$fn:property**

    Define binding to get data from a PM Property.

    Possible syntax:
    - $fn:property(AppletName, PropertyName)
    - $fn:property($, PropertyName)
    - $fn:property(#1, PropertyName)

- **Longform Syntax**

  Use longform syntax when the binding requires additional configuration, multiple inputs, or composition logic that cannot be expressed conveniently through shorthand syntax. In longform syntax, the binding is represented as an object, and the $type property identifies the function that the framework must evaluate.

  Longform syntax is useful when you want to build expressions by combining values, passing parameters, mapping arrays, or invoking functions that require named arguments.

  - **$type**

    Use the $type property to identify the binding function that the framework must evaluate for a longform expression. The value of $type determines how the remaining properties in the object are interpreted.

    Definition of $type by using a join function:

    ```
    {
     "$type": "$fn:join",
     "value": [
     "$fn:value($var:Applet, Account Name, DisplayName)",
     "$fn:value($var:Applet, Account Name, Value)"
     ],
     "combiner": ": "
    }
    ```

    The framework evaluates the object according to the function specified in $type.

  - **$fn:param**

    Use $fn:param to access a parameter value that is passed into a reusable component definition or function context. This binding is typically used inside componentDef so that a reusable component can consume values supplied by the caller.

    Possible syntax:

    - **$fn:param(ParameterName)**

      Example:

      ```
      {
       "componentDef": {
       "ReusableComponent": {
       "params": ["data"],
       "config": {
       "component": "div",
       "children": [
       "$fn:param(data)"
       ]
       }
       }
       }
      }
      ```

- **$fn:object**

  Use $fn:object to build a dynamic object value within the JSON configuration. This binding is useful when the framework must construct an object at run time by resolving one or more bound values instead of using a fully static literal object.

  In this syntax, the framework evaluates the values that are defined in the object and returns the resulting object for use in the configuration.

  Possible syntax:

  - **$fn:object(ObjectDefinition)**

    Longform example:

    ```
    {
     "$type": "$fn:object",
     "value": {
     "label": "$fn:value($, Account Name, DisplayName)",
     "value": "$fn:value($, Account Name, Value)"
     }
    }
    ```

    In this example, the framework evaluates the bound values and returns an object that contains the resolved label and value.

- **$fn:array**

  Use $fn:array to build a dynamic array value within the JSON configuration. This binding is useful when the framework must construct an array at run time by resolving one or more bound values instead of using a fully static literal array.

  In this syntax, the framework evaluates the values that are defined in the array and returns the resulting array for use in the configuration.

  Possible syntax:

  - **$fn:array(ArrayDefinition)**

    Longform example:

    ```
    {
     "$type": "$fn:array",
     "value": [
     "$fn:value($, FirstName, Value)",
     "$fn:value($, LastName, Value)",
     "$fn:value($, Email, Value)"
     ]
    }
    ```

    In this example, the framework evaluates the bound values and returns an array that contains the resolved values.

- **$fn:map**

  Use $fn:map to evaluate one or more values and transform them by using a mapper function. This binding is useful when the rendered output must be derived from the source value instead of displayed directly.

  Possible syntax:

  - **$fn:map(valueArray, mapperFunction)**

    Longform example:

```
{
 "$type": "$fn:map",
 "value": [
 "$fn:value($, FirstName, Value)",
 "$fn:value($, LastName, Value)"
 ],
 "mapper": "$fn:execute($, BuildDisplayName)"
}
```

    This example evaluates multiple values and passes them to the mapper so that the final rendered value can be derived from the source data.

    $fn:map can also accept multiple input values. In this case, the framework evaluates each value in the value array and passes the resolved values to the mapper function so that the final output can be derived from multiple inputs.

    Example:

```
{
 "$type": "$fn:map",
 "value": [
 "$fn:value($, City, Value)",
 "$fn:value($, State, Value)",
 "$fn:value($, Postal Code, Value)"
 ],
 "mapper": "$fn:execute($, BuildAddress)"
}
```

    In this example, the mapper uses multiple resolved values to derive the final rendered output.

- **$fn:join**

  Use $fn:join to combine multiple values into a single rendered string by using a delimiter or separator. This binding is useful when the displayed value must be composed from multiple fields.

  Possible syntax:

  - **$fn:join(valueArray, combiner)**

    Longform example:

```
{
 "$type": "$fn:join",
 "value": [
 "$fn:value($var:Applet, Account Name, DisplayName)",
 "$fn:value($var:Applet, Account Name, Value)"
 ],
 "combiner": ": "
```

```
}
```

This example joins the display name and the value into a single string.

- **$fn:execute**

  Use $fn:execute to invoke a method in the Presentation Model and use the returned value in the JSON configuration. This binding is useful when the rendered value must be computed through custom logic instead of being taken directly from a control or property.

  Possible syntax:

  - **$fn:execute(AppletName, MethodName)**
  - **$fn:execute($, MethodName)**

    You can use $fn:execute(...) directly, or as part of another longform binding such as $fn:map.

Example on Dynamic Data

The following table provides JSON configuration samples:

Description	JSON Config	Output
Joining DisplayName and Value of Control	`{ "$type": "$fn:join", "value": ["$fn:value($var:Applet, Account Name, DisplayName)", "$fn:value($var:Applet, Account Name, Value)"], "combiner": ": " }`	Account Name: Acme Corp
Joining City, State, and Postal Code	`{ "$type": "$fn:join", "value": ["$fn:value($var:Applet, City, Value)", "$fn:value($var:Applet, State, Value)", "$fn:value($var:Applet, Postal Code, Value)"], "combiner": ", " }`	San Francisco, CA, 94105
Mapping Status value to badge color	`{ "$type": "$fn:map", "value": ["$fn:value($var:Applet, Status, Value)"], "mapper": "$fn:execute($var:Applet, MapStatusToColor)" }`	danger or warning or another mapped value
Accessing a parameter value by using $fn:param	`"$fn:param(data['Order Number'])"`	1-ABC123
Mapping multiple values by using map	`{ "$type": "$fn:map", "value": ["$fn:value($, FirstName, Value)", "$fn:value($, LastName, Value)"], "mapper": "$fn:execute($, BuildDisplayName)" }`	John Smith
Build Object	`{ "$type": "$fn:map",`	Object output returned by the mapper

Description	JSON Config	Output
	`"value": ["$fn:value($, Account Name, Value)", "$fn:value($, Location, Value)"], "mapper": "$fn:execute($, BuildObject)" }`	
Build Array	`{ "$type": "$fn:map", "value": ["$fn:value($, Phone, Value)", "$fn:value($, Email, Value)"], "mapper": "$fn:execute($, BuildArray)" }`	Array output returned by the mapper
Mix-and-match	`Combination example that uses shorthand and longform bindings together in the same JSON definition`	Mixed rendered output
Complex One	`Composite example that combines $fn:join, $fn:map, $fn:param, and $fn:execute in one configuration`	Derived composite output

Note: Existing shorthand bindings continue to be supported. Use longform syntax when the binding requires named properties, composition logic, mapper functions, or reusable parameterized configuration. For new JSON definitions, prefer the syntax that best balances readability and functional clarity for the use case.

Customizing Using Oracle JET Components in the Web Component Framework

This topic describes examples of how the Web Component Framework uses a presentation model (PM) and the Oracle JET Component of an applet view and a few other layouts to render the components on the UI.

- *Configuring a Form Layout*
- *Configuring a List Layout*
- *Configuring a Chart Layout*
- *Configuring a Tree Layout*
- *Configuring a View Layout*
- *Configuring a Foldout Layout*

Configuring a Form Layout

The framework provides the capability to render a responsive form layout for a Siebel form applet based on JSON configuration embedded within a custom Presentation Model (PM). The JSON configuration uses the configuration options that are supported by the Oracle JET oj-c-form-layout component.

The framework also provides additional attributes that help bind Siebel applet controls to the configured layout. Use this configuration to define the structural layout of the form, while the rendering of individual controls continues to follow the corresponding Siebel control configuration.

The detailed documentation for the Oracle JET form layout component is available in the Oracle JET documentation. The following examples describe common customization use cases for configuring a form layout.

To configure a form layout:

1. Create a custom Presentation Model (PM) file for the Siebel form applet. The custom PM must extend FormModelViewPM, which provides the base framework behavior for rendering a Siebel form applet through the Web Component Framework.

```
define('siebel/simpleformlayoutpm', ['siebel/formmodelviewpm'], function () {
 SiebelJS.Namespace("SiebelAppFacade.SimpleFormLayoutPM");
 SiebelAppFacade.SimpleFormLayoutPM = (function () {
 function SimpleFormLayoutPM() {
 SiebelAppFacade.SimpleFormLayoutPM.superclass.constructor.apply(this, arguments);
 }

 SiebelJS.Extend(SimpleFormLayoutPM, SiebelAppFacade.FormModelViewPM);

 SimpleFormLayoutPM.prototype.Init = function () {
 SiebelAppFacade.SimpleFormLayoutPM.superclass.Init.apply(this, arguments);
 };

 return SimpleFormLayoutPM;
 }());

 return "SiebelAppFacade.SimpleFormLayoutPM";
});
```

2. Go to the **Manifest Administration** view to configure the manifest.
   a. Create a new entry in Applet "UI Objects":
      - Type: Applet
      - Usage Type: Presentation Model
      - Name: <Applet Name>
   b. Create a record in "Object Expression" applet:
      - Expression: "Redwood Theme"
      - Level: 1

   For the above record, associate file "siebel/simpleformlayoutpm" with it by using the Applet "Files".
3. Review the JSON configuration. The following examples describe common customization use cases.

   For more information, see the detailed documentation for the Oracle JET form layout component. The framework also provides additional attributes for binding with Siebel applet controls.

   SiebelAppFacade.FormModelViewPM is the base JavaScript Presentation Model (PM) class for a Siebel form applet that is used with an Oracle JET component. Any custom PM must extend

SiebelAppFacade.FormModelViewPM and declare dependency on siebel/formmodelviewpm as part of the define statement.

a. Use All Applet Controls

   Provide the following JSON specification to render a responsive form layout by using all applet controls that are mapped to the form applet:

```
{
 "reuseHeaderFooter": true,
 "maxColumns": 3,
 "direction": "row",
 "labelEdge": "inside",
 "userAssistanceDensity": "compact"
}
```

   The reuseHeaderFooter property is a Siebel-specific property. If it is set to true, then the header and footer template are reused and rendered through FormTemplateRenderer. If it is set to false, then the full form container is replaced.

   The properties maxColumns, direction, labelEdge, and userAssistanceDensity are configuration attributes of the Oracle JET oj-c-form-layout component.

   With this JSON configuration, the framework renders all applet controls that are configured for the applet in a responsive three-column layout.

b. Selective Controls or Client-side Layout Order Rendering

   The framework provides the layout attribute in the JSON specification to define the order in which the framework must place controls in the form layout. Use this attribute when you want to render only selected applet controls or when you want to control the client-side layout order explicitly.

   For example, the following JSON specification renders the applet controls that are identified through the field attribute in a responsive three-column form layout:

```
{
 "maxColumns": 3,
 "direction": "row",
 "labelEdge": "inside",
 "layout": [
 { "field": "Name" },
 { "field": "Designation" },
 { "field": "Email" },
 { "field": "Phone #" }
 ]
}
```

   The layout attribute accepts an array of objects. Each object in the array can map either to an applet control or to a nested form layout. When the type attribute is set to control, the field attribute is mapped to the corresponding Siebel applet control. When the type attribute is set to form, the object defines a nested form layout.

   The default value of the type attribute is control. Therefore, you can omit the type attribute when the entry maps directly to an applet control.

c. Expand Applet Control across columns

The Oracle JET oj-c-form-layout component provides the capability to span a user interface control across multiple columns. Use the columnSpan attribute when a control must occupy more than one column in the responsive form layout.

For example, the following JSON configuration indicates that the applet control Name must span two columns in a responsive two-column layout. In this case, the second row of the layout contains the Designation and Email controls.

```
{
 "maxColumns": 2,
 "direction": "row",
 "labelEdge": "inside",
 "layout": [
 { "field": "Name", "columnSpan": 2 },
 { "field": "Designation" },
 { "field": "Email" }
 ]
}
```

d. Nested Form Layout

Use a nested form layout when a section of the overall form must be rendered as a separate form structure within the parent layout. In this case, set the type attribute to form and define a nested layout array for that section.

The following example shows a responsive form layout that contains nested form sections:

```
{
 "maxColumns": 3,
 "direction": "row",
 "labelEdge": "inside",
 "userAssistanceDensity": "compact",
 "layout": [
 { "field": "M/M" },
 { "field": "Full Name Title", "columnSpan": 2 },
 {
 "type": "form",
 "columnSpan": 2,
 "direction": "column",
 "layout": [
 { "field": "Street Address" },
 { "field": "City" },
 { "field": "State" },
 { "field": "Postal Code" }
 ]
 },
 {
 "type": "form",
 "layout": [
 { "field": "Work Phone #" },
 { "field": "Fax #" }
 ]
 }
 ]
```

```
}
```

In this example, each object with type set to form defines a nested form layout within the parent form. This approach lets you create more complex responsive form structures while continuing to use the same JSON layout model.

It is recommended that the JSON configuration use index-based mapping as much as possible so that a form layout can be reused for similar configurations. The JSON configuration defines only the structural layout of the responsive form. The rendering of each control depends on the Siebel control configuration and the control type that is mapped at run time.

e. Simple Form Section

Use a simple form section when the form layout must group related controls into separate sections within the same responsive form. This approach is useful when the user interface must organize controls into functional sections while keeping the configuration easy to read and maintain.

```
{
 "maxColumns": 4,
 "direction": "row",
 "userAssistanceDensity": "compact",
 "layout": [
 {
 "type": "form",
 "columnSpan": 2,
 "maxColumns": 2,
 "layout": [
 { "field": "HTML Label 2", "columnSpan": 2, "uiType": "Label" },
 { "field": "SRNumber" },
 { "field": "Description" }
 ]
 },
 {
 "type": "form",
 "columnSpan": 2,
 "maxColumns": 2,
 "layout": [
 { "field": "Status" },
 { "field": "Priority" },
 { "field": "Owner" }
 ]
 }
 ]
}
```

In this example, each section is defined as a nested form layout within the parent form. This configuration helps organize related controls into separate sections while continuing to use the same responsive layout model.

f. Multilevel Form Section

Use a multilevel form section when the form layout must contain multiple nested form sections within the parent form. This approach is useful when the user interface must group related controls into separate logical sections while still preserving a responsive overall layout.

The following example shows a multilevel form section configuration:

```
{
 "maxColumns": 4,
 "direction": "row",
 "userAssistanceDensity": "compact",
 "layout": [
```

```
{
"type": "form",
"columnSpan": 2,
"maxColumns": 2,
"layout": [
{ "field": "HTML Label 2", "columnSpan": 2, "uiType": "Label" },
{ "field": "SRNumber" },
{
"type": "form",
"columnSpan": 2,
"maxColumns": 2,
"layout": [
{ "field": "Status" },
{ "field": "Priority" }
]
}
]
},
{
"type": "form",
"columnSpan": 2,
"maxColumns": 2,
"layout": [
{ "field": "Owner" },
{ "field": "Organization" }
]
}
]
}
```

In this example, the parent form contains multiple nested form sections, and some of those sections contain additional nested form layouts. This configuration helps organize complex form content into logical groups while continuing to use the same JSON-based layout model.

Configuring a List Layout

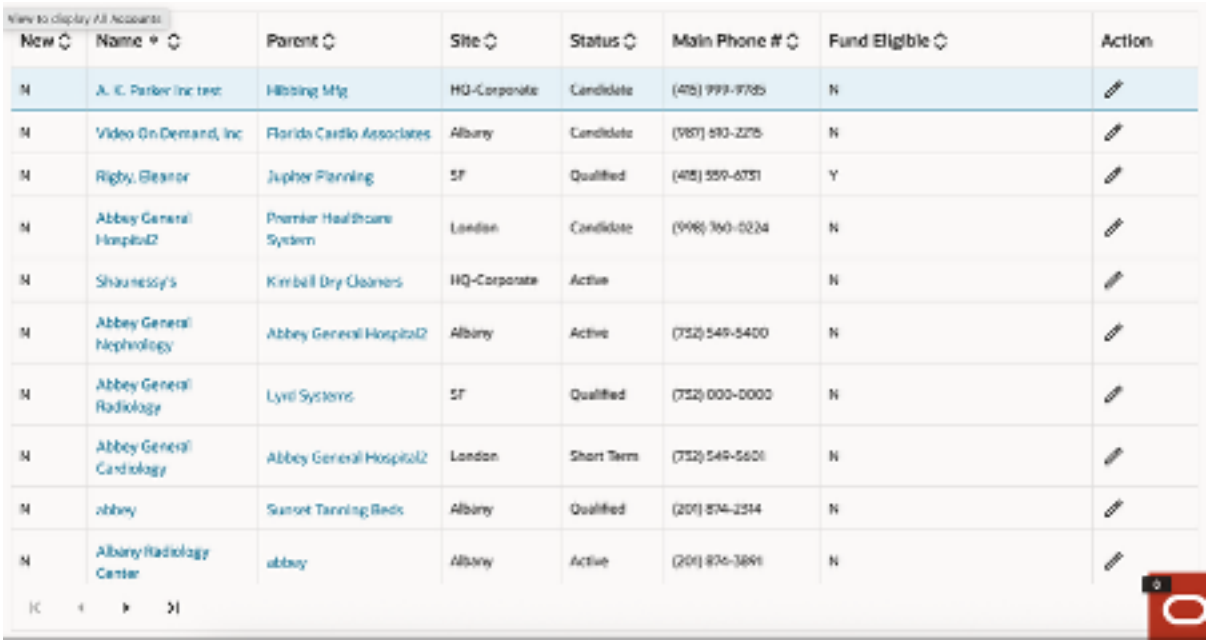
The framework provides the capability to render a table for a Siebel list applet or hierarchical list applet based on JSON configuration embedded in a custom Presentation Model (PM). To render the list, the framework uses the Oracle JET oj-table component. Therefore, the JSON configuration can use the configuration options that the Oracle JET oj-table component supports, along with Siebel-specific extensions.

Use this configuration to define table properties, data-column properties, non-data-column properties, and the overall column sequence for the rendered list. The framework also provides additional attributes and behaviors for binding Siebel list applet metadata and data to the configured layout.

The following examples describe common customization use cases for configuring a list layout.

You can configure siebel/listmodelviewpm as the Presentation Model (PM) for the corresponding list applet to render the list or hierarchical list by using the Oracle JET oj-table component. In this case, all column controls that are configured for the list and exposed through the related applet web template are rendered through the framework.

Here is a sample screenshot of how UI would appear for such use case:



To configure the default list layout:

1. Create a custom Presentation Model (PM) file for the Siebel list applet. The custom PM must extend ListModelViewPM, which provides the base framework behavior for rendering a Siebel list applet through the Web Component Framework.

```
define('siebel/mycustomlistpm', ['siebel/listmodelviewpm'], function () {
 "use strict";

 SiebelJS.Namespace("SiebelAppFacade.MyCustomListPM");
 SiebelAppFacade.MyCustomListPM = (function () {
 function MyCustomListPM() {
 SiebelAppFacade.MyCustomListPM.superclass.constructor.apply(this, arguments);
 }

 SiebelJS.Extend(MyCustomListPM, SiebelAppFacade.ListModelViewPM);

 MyCustomListPM.prototype.Init = function () {
 SiebelAppFacade.MyCustomListPM.superclass.Init.apply(this, arguments);
 };

 return MyCustomListPM;
 }());

 return "SiebelAppFacade.MyCustomListPM";
});
```

2. Go to the **Manifest Administration** view to configure the manifest.

   a. Create a new entry in Applet "UI Objects":

      - Type: Applet
      - Usage Type: Presentation Model
      - Name: <Applet Name>

   b. Create a record in "Object Expression" applet:
      - Expression: "Redwood Theme"
      - Level: 1

   For the above record, associate file "siebel/mycustomlistpm" with it by using the Applet "Files".

3. Review the JSON configuration

   a. Default List Layout.

      The following example shows a basic JSON configuration for rendering a list applet through the Oracle JET oj-table component:

```
{
 "editMode": "rowEdit",
 "horizontalGridVisible": "enabled",
 "verticalGridVisible": "enabled"
}
```

      This configuration applies Oracle JET table properties at the top level of the JSON definition.

   b. Data Columns Sequence Controlled by ODH Template with Customization.

      The framework provides the capability to render a Siebel list applet by using JSON configuration embedded in a custom Presentation Model (PM). Use this approach when you want to retain the column sequence from the ODH template but override selected table properties or selected column properties.

      Use this configuration to:

      - Retain the columns configured in the ODH template, while honoring the overriding configuration that the JSON provides.
      - Override selected table properties or selected data-column properties without redefining the entire list layout.

      JSON configuration for this scenario:

```
{
 "editMode": "rowEdit",
 "horizontalGridVisible": "enabled",
 "verticalGridVisible": "enabled",
 "columns": [
 {
 "field": "Parent Account Name",
 "resizable": "disabled",
 "templates": {
 "Edit": {
 "UIType": "JText"
 }
 }
 }
 ]
```

```
}
```

The JSON configuration provides configuration at two levels. At the first level, it defines Oracle JET table properties that oj-table supports. At the second level, it defines data-column or non-data-column properties in the columns property of the Oracle JET table.

For example, the following JSON defines a data-column override:

```
{
 "field": "Parent Account Name",
 "resizable": "disabled",
 "templates": {
 "Edit": {
 "UIType": "JText"
 }
 }
}
```

Use this configuration when you want to override the ODH template column order and render only a selected subset of columns. In this case, the framework renders only the columns that the JSON configuration defines and honors the order in which those columns appear in the columns property.

To support this behavior, set the top-level JSON property columnsSequenceOverride to true.

JSON configuration for this scenario:

```
const JSON_PROPERTY_VALUE = {
 columnsSequenceOverride: true,
 editMode: "rowEdit",
 horizontalGridVisible: "enabled",
 columns: [
 {
 field: "Row Status"
 },
 {
 field: "Name",
 headerText: "Account Name"
 },
 {
 field: "Parent Account Name",
 resizable: "disabled",
 templates: {
 Edit: {
 UIType: "JText"
 }
 }
 }
 ]
};
```

In this configuration, only the columns listed in the JSON are rendered, and they are displayed in the same sequence in which they appear in the columns array.

4. Convert Oracle JET kebab-case property names to camelCase in the JSON configuration. For example, vertical-grid-visible becomes verticalGridVisible, and selection-mode becomes selectionMode.
5. If a property accepts an object value, provide the value in JSON object form according to the Oracle JET property definition. For example, the selectionMode property can contain keys such as row or column, depending on the Oracle JET documentation.

Multi-select Support

- selection-mode.row='multiple' (default), the row selection checkbox column can be displayed.

- The row selection checkbox column is always displayed when "MultiSelect" is enabled for a list applet. The Open UI applet property Multi Row Select Checkbox Display is no longer honored based on device capability.
- To customize multi-select behavior, use the applet (SWEFrame) method MultiSelect. Set the CanInvoke value for this method to true or false, or set selection-mode.row='multiple' or selection-mode.row='single'.

> **Note:** Unsupported Open UI features must not be assumed to work automatically in the Web Component Framework list layout. Only the controls, bindings, and extensions that are documented for this framework are supported. If a specific Open UI behavior depends on legacy rendering logic, then additional redesign or customization might be required.

Configuring a Chart Layout

The framework provides the capability to render a Siebel chart applet by using a functional Presentation Model (PM) that interfaces with the Oracle JET oj-chart component. By using the capabilities of oj-chart, the framework renders a modern, responsive chart visualization.

The objective of this configuration is to provide a functional Presentation Model (PM) component that interfaces with the Oracle JET oj-chart component to render a Siebel chart applet.

This layout provides a modern, responsive chart-based visualization UI, implements a Presentation Model (PM) wrapper that manages chart data, series (YAxisData), and groups (XAxisData), uses the Siebel PM for dynamic data binding and event handling while exposing formatted chart data to the Oracle JET chart component, supports bar, line, and pie Oracle JET chart types, and enables integration of color palettes, legends, and tooltips through Oracle JET chart configuration.

SiebelAppFacade.ChartModelView is the base JavaScript Presentation Model (PM) class for a Siebel chart applet that is used with an Oracle JET component. Any custom PM must extend SiebelAppFacade.ChartModelView and declare a dependency on siebel/chartmodelviewpm as part of the define statement.

Following functions are supported:

- Dynamic data binding from Siebel business component data to Oracle JET chart series and groups.
- Multiple chart types, such as bar, line, and pie.
- Legend and tooltip support.
- Color attribute groups support through Oracle JET ColorAttributeGroupHandler.
- Responsive layout.
- Chart orientation. Horizontal indicates that the chart is horizontally oriented. Vertical indicates that the chart is vertically oriented, which is the default. This applies only to bar and line charts.

To configure a Chart Layout:

1. Configure the applet by following the standard configuration steps for a chart applet. For more information, see Siebel Special Purpose Applets Guide.

2. Go to the **Manifest Administration** view to configure the manifest.
    a. Create a new entry in Applet "UI Objects":
        - Type: Applet
        - Usage Type: Presentation Model
        - Name: <Applet Name>
    b. Create a record in the "Object Expression" applet:
        - Expression: "Redwood Theme"
        - Level: 1

    For the preceding record, associate the file `siebel/chartmodelview` with it by using the Applet Files.

Validation

Validate the chart user interface to confirm correct rendering, axes, legends, and tooltip behavior. Also confirm that chart-related event handling works as expected.

Test the chart in Siebel UI to ensure:

- Data binding to chart series and groups works correctly.
- Under the Redwood theme, the applet displays Oracle JET chart behavior as expected.

Customization Scenario

- Extend chart configuration to support additional Oracle JET chart options, such as animation, zooming, and stack-related options.
- For advanced customization, see Siebel Special Purpose Applets Guide.

Configuring a Tree Layout

The framework provides the capability to render a Siebel tree applet by using a functional Presentation Model (PM) that interfaces with the Oracle JET oj-tree-view component. The framework-provided PM handles dynamic data binding and event handling for the Oracle JET tree component and supports expand and collapse operations.

SiebelAppFacade.TreeModelView is the base JavaScript Presentation Model (PM) class for a Siebel tree applet that is used with an Oracle JET component. Any custom PM must extend SiebelAppFacade.TreeModelView and declare a dependency on siebel/treemodelviewpm as part of the define statement.

You can configure siebel/treemodelviewpm as the Presentation Model (PM) for the corresponding tree applet to render the tree by using the oj-tree-view component. More component-based customization capabilities are not available with this release.

This implementation supports dynamic data binding between Siebel data and the Oracle JET tree structure, expand and collapse behavior, selection handling that is consistent with supported Siebel interaction behavior, responsive layout, and support for custom node icons through supported Siebel or Oracle JET mechanisms.

To configure a Tree Layout:

1. Configure the applet by following the standard configuration steps for a tree applet.
2. Go to the **Manifest Administration** view to configure the manifest.
    a. Create a new entry in Applet "UI Objects":
        - Type: Applet
        - Usage Type: Presentation Model

- Name: <Applet Name>

b. Create a record in the "Object Expression" applet:

- Expression: "Redwood Theme"
- Level: 1

For the preceding record, associate the file `siebel/treemodelviewpm` with it by using the Applet Files.

Validation

Validate the tree user interface to confirm correct rendering, icon display, and expand and collapse behavior. Also confirm that tree-related event handling works as expected.

Test the tree in Siebel UI to ensure:

- The tree renders correctly and all required nodes are displayed.
- Expand and collapse behavior works correctly.
- Selection behavior works as expected.
- Under the Redwood theme, the applet displays Oracle JET tree behavior as expected.

Customization Scenario

- You can apply custom icons or images for individual node items and extend styling through CSS or supported icon mapping mechanisms, such as nodeIconsMap, where applicable.
- For advanced customization, see *Siebel Special Purpose Applets Guide*.

Known Issues

- Single selection only, by default.
- Limited icon sources, such as Siebel nodeIconsMap or supported Oracle JET helper class CSS.

Configuring a View Layout

Similar to applets, Siebel views can be configured to render by using Oracle JET or Spectra components through a custom Presentation Model (PM). In this approach, the JSON specification is provided through the PM property SEBL_COMPONENT_CONFIG.

SiebelAppFacade.BaseViewModelViewPM is the base JavaScript Presentation Model (PM) class for a Siebel view that is used with an Oracle JET component. Any custom PM must extend SiebelAppFacade.BaseViewModelViewPM and declare a dependency on siebel/baseviewmodelviewpm as part of the define statement.

Traditionally, Open UI applet controls can be rendered only within the applet boundary. By using this framework, that limitation is removed, and applet controls can be rendered anywhere in the view.

Solution Overview

The framework provides a declarative JSON approach in which the developer defines the bindings between Siebel Open UI and Oracle JET or Spectra components, the HTML structure that the Oracle JET or Spectra component expects, and the JSON structure that represents the markup and attribute definitions of the component. The framework then builds the HTML markup and generates the bindings.

Key Features

- Reduced boilerplate, because most rendering can be defined through JSON instead of custom JavaScript.

- Out-of-the-box integration with Siebel Open UI, including applets, applet controls, Presentation Model (PM) methods, and PM properties.
- Support for reusing existing applets, whether they are defined through this JSON framework, implemented through JavaScript, or provided out of the box.
- Support for value manipulation functions within the JSON specification.
- Support for variables for reuse and simplification of configuration.

Examples:

- Mini Label and Value Pair

  The following example shows how to render the display name and value of a Siebel control in a simple view-level structure.

```
{
 "imports": [],
 "variables": {
 "Applet": "SIS Account List Applet"
 },
 "component": "div",
 "children": [
 "$fn:value($var:Applet, Account Name, DisplayName)",
 " : ",
 {
 "component": "b",
 "children": [
 "$fn:value($var:Applet, Account Name, Value)"
 ]
 }
 ]
}
```

  This example uses shorthand $fn:value(...) bindings for both DisplayName and Value.

- Simple Form by Using oj-c-input-text and Variables

  The following example demonstrates how variables can be used to simplify view-level configuration. This example is not the recommended approach for rendering a full form, because FormModelViewPM is available for that purpose. It is included here to demonstrate the flexibility of the framework.

```
{
 "imports": [
 "oj-c/input-text",
 "oj-c/form-layout"
 ],
 "variables": {
 "Applet": "Account Address Applet",
 "width": "md:6"
 },
 "component": "oj-c-form-layout",
 "attributes": {
 "direction": "row"
 },
 "children": [
 {
 "component": "oj-c-input-text",
 "attributes": {
 "label-hint": "$fn:value($var:Applet, Street Address, DisplayName)",
 "value": "$fn:value($var:Applet, Street Address, Value)",
 "class": "$var:width"
 }
```

```
 },
 {
 "component": "oj-c-input-text",
 "attributes": {
 "label-hint": "$fn:value($var:Applet, City, DisplayName)",
 "value": "$fn:value($var:Applet, City, Value)",
 "class": "$var:width"
 }
 },
 {
 "component": "oj-c-input-text",
 "attributes": {
 "label-hint": "$fn:value($var:Applet, Postal Code, DisplayName)",
 "value": "$fn:value($var:Applet, Postal Code, Value)",
 "class": "$var:width"
 }
 }
 ]
}
```

This example demonstrates variable substitution through $var: and binding to applet controls through $fn:value(...).

- Foldout Layout with Embedded Applets

  The following example shows how a view layout can reuse existing Siebel applets within a Spectra foldout layout.

```
{
 "imports": [
 "oj-sp/foldout-layout/loader",
 "oj-sp/foldout-panel-summarizing/loader"
 ],
 "component": "oj-sp-foldout-layout",
 "attributes": {
 "default-expand": "false"
 },
 "children": [
 {
 "component": "oj-sp-foldout-panel-summarizing",
 "attributes": {
 "summary": "Account Info",
 "illustration": "oj-ux-ico-contact"
 },
 "children": [
 {
 "component": "$Applet(Account Entry Applet)"
 }
 ]
 },
 {
 "component": "oj-sp/foldout-panel-summarizing",
 "attributes": {
 "summary": "Recent Orders",
 "illustration": "oj-ux-ico-order"
 },
 "children": [
 {
 "component": "$Applet(Order Entry - Line Items List Applet)"
 }
 ]
 }
 ]
```

```
}
```

This example shows how existing applets can be embedded as child elements within a view-level component layout.

- Horizontal Overview by Using join and map

  The following example shows how to build a view-level overview layout by using longform functions such as join and map.

```
{
 "imports": [
 "oj-sp/horizontal-overview/loader",
 "oj-sp/empty-state/loader"
 ],
 "variables": {
 "statusMapMethod": "$fn:execute($, MapStatusToColor)"
 },
 "component": "oj-sp-horizontal-overview",
 "attributes": {
 "layout": "wrap"
 },
 "children": [
 {
 "$type": "$fn:join",
 "value": [
 "$fn:value($, Account Name, Value)",
 "$fn:value($, Status, Value)"
 ],
 "combiner": " - "
 },
 {
 "$type": "$fn:object",
 "value": {
 "component": "oj-badge",
 "attributes": {
 "color": {
 "$type": "$fn:map",
 "value": "$fn:value($, Status, Value)",
 "mapper": "$var:statusMapMethod"
 },
 "label": "$fn:value($, Status, Value)"
 }
 }
 },
 {
 "component": "oj-button",
 "attributes": {
 "label": "Open Record",
 "on-oj-action": "$fn:execute($, OpenRecord)"
 }
 }
 ]
}
```

This example demonstrates the use of longform functions to compose values and derive presentation attributes dynamically.

- Recommendation Card

  The following example shows how static values, variables, and dynamic bindings can be combined in a Spectra recommendation card.

```
{
 "imports": [
 "oj-sp/recommendation-card/loader"
 ],
 "variables": {
 "color": "neutral"
 },
 "component": "oj-sp-recommendation-card",
 "attributes": {
 "cardTitle": "$fn:value($, cardTitle, DisplayName)",
 "description": "$fn:value($, Description, DisplayName)",
 "displayOptions": {
 "illustration": "on"
 },
 "recommendedAction": {
 "$type": "$fn:object",
 "value": {
 "label": "$fn:value($, Pay Bill, DisplayName)"
 }
 },
 "onspRecommendedAction": "$fn:execute($, Pay Bill)",
 "themeMode": "$var:color",
 "backgroundColor": "$var:color",
 "layout": "fitContent"
 },
 "children": []
}
```

  This example demonstrates the use of variables, static attributes, and dynamic bindings in a view-level component configuration.

  You can also place object definitions in the variables section and reference them by using $var: when reuse is required.

Nested Function Call

Function calls can be nested in the JSON specification. For example:

```
$fn:value($, $fn:property($, Property1), Value)
```

In this example, the control name that is passed as the second argument to $fn:value(...) is resolved from the PM property Property1.

> **Note:** If an applet name, control name, or property contains an unbalanced parenthesis and must be used in a $fn: expression, define that value in the variables section and reference it through $var:

Component Registration and Use in JSON Configuration

A reusable component that is registered in the ModelViewFactory can be referenced from the JSON configuration.

Use the following syntax to reference a registered component:

```
{
 "component": "$fn:component(TestModelView)",
 "attributes": {
 "pm": "$fn:Applet($)",
```

```
 "config": {}
 }
}
```

Note: Earlier configurations might use $WebComp:(TestModelView) to reference a registered component. Use $fn:component(TestModelView) for new configurations.

Consolidating in the PM

1. Create a custom PM that extends SiebelAppFacade.BaseViewModelViewPM.
2. Set the SEBL_COMPONENT_CONFIG property in the Init method.

```
define('siebel/custom/customviewpm', ['siebel/baseviewmodelviewpm'], function () {
 SiebelJS.Namespace('SiebelAppFacade.CustomViewPM');
 SiebelAppFacade.CustomViewPM = (function () {

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

 SiebelJS.Extend(CustomViewPM, SiebelAppFacade.BaseViewModelViewPM);

 CustomViewPM.prototype.Init = function () {
 SiebelAppFacade.CustomViewPM.superclass.Init.apply(this, arguments);

 var JSON_CONFIG = {
 "imports": [],
 "container": "",
 "variables": {},
 "component": "...",
 "attributes": {},
 "children": []
 };

 this.AddProperty("SEBL_COMPONENT_CONFIG", JSON_CONFIG);
 };

 return CustomViewPM;
 }());

 return SiebelAppFacade.CustomViewPM;
});
```

Manifest Configuration

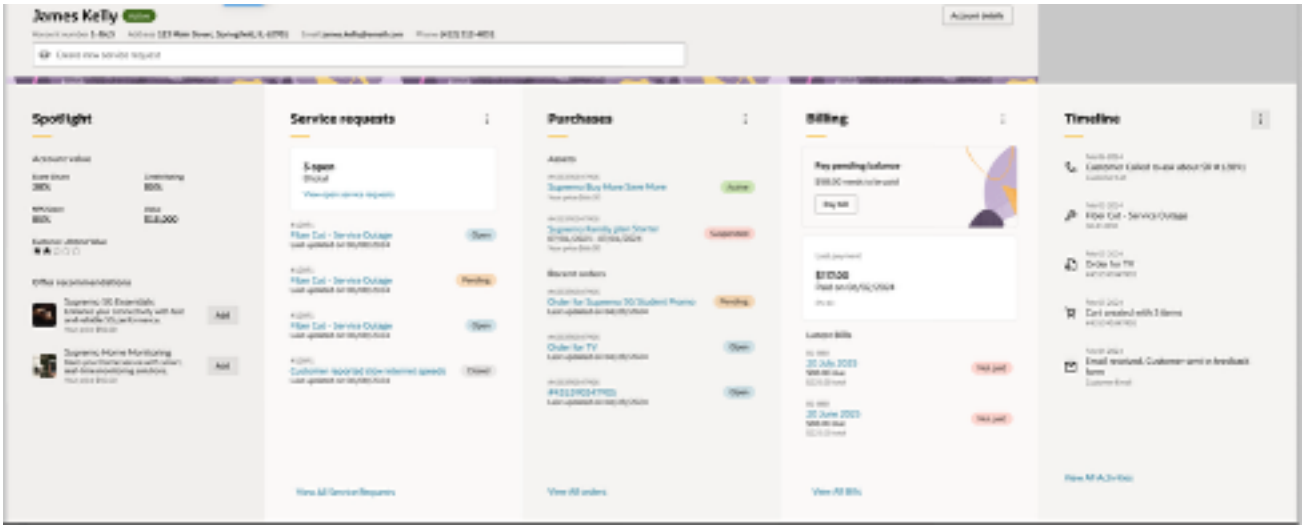
The view or applet, depending on what configuration the developer is writing, must be configured in the manifest so that the correct custom PM is loaded.

Configuring a Foldout Layout

The Web Component Framework provides the capability to render a Siebel view by using the Oracle JET oj-sp-foldout-layout component. This layout is useful when you want to present an overview region together with multiple foldout panels that organize related applet content in a modern, responsive user interface.

The Foldout Layout configuration is defined in the `SEBL_COMPONENT_CONFIG` property of a custom view Presentation Model (PM). The JSON specification defines the imported Oracle JET modules, the variables used across the configuration, the main foldout layout component, and the child components that are rendered in the supported slots.

Sample UI :



Following functions are supported:

- A horizontal overview region in the overview slot.
- Multiple oj-sp-foldout-panel-summarizing panels in the default slot.
- Dynamic binding of panel title, action label, and action state to Siebel applet controls.
- Secondary actions by using supported JSON functions such as $fn:array, $fn:object, $fn:join, $fn:map, and $fn:execute.
- Optional content, such as recommendation content, inside supported panel slots.
- Reuse of existing Siebel applets inside the foldout layout.

To configure a Foldout Layout:

1. Create a custom view PM that extends SiebelAppFacade.BaseViewModelViewPM.
2. Define the Foldout Layout JSON in the SEBL_COMPONENT_CONFIG property.
3. Configure the overview area and the foldout panels in the children array.
4. Configure the manifest so that the custom PM is loaded for the view.
5. Configure any applets that are reused in the overview region or foldout panels by using their corresponding PMs.

Example of Foldout Layout JSON

```
{
 "imports": [
 "ojs/ojinputsearch",
 "oj-sp/foldout-layout/loader",
 "oj-sp/foldout-panel-summarizing/loader",
 "oj-sp/empty-state/loader",
 "oj-sp/horizontal-overview/loader",
 "ojs/ojlabelvalue",
 "oj-c/input-text",
 "ojs/ojformlayout"
 ],
 "variables": {
```

```
"Applet1": "<Applet Name #1>",
"Applet2": "<Applet Name #2>",
"Applet3": "<Applet Name #3>",
"Applet4": "<Applet Name #4>",
"Applet5": "<Applet Name #5>",
"Applet6": "<Applet Name #6>",
"Applet7": "<Applet Name #7>",
"AppletTitle": "<Siebel control used for the panel title>",
"ViewAll": "<Siebel control used for the View All action>",
"PayBill": "<Siebel control used for a panel action>"
},
"component": "oj-sp-foldout-layout",
"attributes": {
"orientation": "horizontal",
"displayMode": "light",
"animate": "auto",
"displayOptions": {
"goToParent": false,
"inFlowBack": true
}
},
"children": [
{
"component": "div",
"attributes": {
"slot": "overview"
},
"children": [
{
"component": "$Applet:($var:Applet1)"
}
]
},
{
"component": "div",
"attributes": {
"slot": "search"
},
"children": [
{
"component": "$Applet:()"
}
]
},
{
"component": "oj-sp-foldout-panel-summarizing",
"attributes": {
"panelTitle": "$fn:value($var:Applet2,$var:AppletTitle,DisplayName)",
"overflowActionLabel": "$fn:value($var:Applet2,$var:ViewAll,DisplayName)",
"onspOverflowAction": "$fn:execute($var:Applet2,$var:ViewAll)",
"secondaryActions": {
"$type": "$fn:array",
"value": [
{
"$type": "$fn:object",
"value": {
"id": {
"$type": "$fn:join",
"value": [
"$var:Applet2",
"$var:AppletTitle"
],
"combiner": ":"
},
"label": "$fn:value($var:Applet2,$var:AppletTitle,DisplayName)",
"icon": "oj-ux-ico-actions",
```

```
"display": {
"$type": "$fn:map",
"value": "$fn:value($var:Applet2,$var:AppletTitle,CanInvoke)",
"mapper": "$fn:execute($,BooleanToOnDisabled)"
}
}
},
{
"$type": "$fn:object",
"value": {
"id": {
"$type": "$fn:join",
"value": [
"$var:Applet2",
"$var:ViewAll"
],
"combiner": ":"
},
"label": "$fn:value($var:Applet2,$var:ViewAll,DisplayName)",
"icon": "oj-ux-ico-actions",
"display": {
"$type": "$fn:map",
"value": "$fn:value($var:Applet2,$var:ViewAll,CanInvoke)",
"mapper": "$fn:execute($,BooleanToOnDisabled)"
}
}
}
]
},
"onspAction": "$fn:execute($,SecondaryActionsHandler)"
},
"children": [
{
"component": "div",
"attributes": {},
"children": [
{
"component": "$Applet:($var:Applet2)"
}
]
}
]
},
{
"component": "oj-sp-foldout-panel-summarizing",
"attributes": {
"panelTitle": "$fn:value($var:Applet3,$var:AppletTitle,DisplayName)",
"overflowActionLabel": "$fn:value($var:Applet3,$var:ViewAll,DisplayName)",
"onspOverflowAction": "$fn:execute($var:Applet3,$var:ViewAll)"
},
"children": [
{
"component": "div",
"attributes": {},
"children": [
{
"component": "$Applet:($var:Applet3)"
}
]
},
{
"component": "div",
"attributes": {
"slot": "recommendation"
},
"children": [
```

```
 {
 "component": "$Applet:($var:Applet7)"
 }
 ]
 }
 ]
 }
 ]
}
```

Configuring the Foldout Layout JSON

1. Add all required Oracle JET modules in the imports array.
2. Define reusable values, such as applet names and control names, in the variables object.
3. Set the main component value to oj-sp-foldout-layout.
4. Define layout-wide settings, such as orientation, displayMode, animate, and displayOptions, in the attributes object.
5. Use the children array to define content for the supported slots.
6. Use the overview slot for the overview applet or component.
7. Use the default slot for one or more oj-sp-foldout-panel-summarizing components.
8. Use supported dynamic bindings to populate panel title, overflow action label, and action state from Siebel data.
9. Add multiple foldout panel JSON objects as required for the business use case.

Overview Slot

The overview slot is typically used to display summary information for the current record or context. The following example shows the JSON structure for the overview slot:

```
{
 "component": "div",
 "attributes": {
 "slot": "overview"
 },
 "children": [
 {
 "component": "$Applet:($var:Applet1)"
 }
 ]
}
```

You can configure the applet used in the overview slot by using a dedicated PM that renders a horizontal overview web component.

Default Foldout Panel

The default slot is used to render one or more oj-sp-foldout-panel-summarizing components. Each panel can display applet content and optional actions.

```
{
 "component": "oj-sp-foldout-panel-summarizing",
 "attributes": {
 "panelTitle": "$fn:value($var:Applet2,$var:AppletTitle,DisplayName)",
 "overflowActionLabel": "$fn:value($var:Applet2,$var:ViewAll,DisplayName)",
 "onspOverflowAction": "$fn:execute($var:Applet2,$var:ViewAll)"
 },
 "children": [
 {
 "component": "div",
 "attributes": {},
 "children": [
 {
```

```
 "component": "$Applet:($var:Applet2)"
 }
 ]
 }
 ]
}
```

This configuration renders the specified applet inside the foldout panel and binds the panel title and overflow action to Siebel controls.

Configuring JSON for the Horizontal Overview Applet

The Foldout Layout often uses a horizontal overview applet in the overview slot. Configure that applet separately by using a dedicated PM. The following example shows a representative JSON structure:

```
{
 "imports": [
 "oj-sp/horizontal-overview/loader",
 "oj-sp/smart-search/loader",
 "ojs/ojlabelvalue",
 "ojs/ojformlayout"
 ],
 "variables": {
 "Applet": "<Applet Name>",
 "itemTitle": "Name",
 "Status": "Status",
 "AccountNumber": "Account Number",
 "StreetAddress": "Address1",
 "EmailAddress": "Main Email Address",
 "PhoneNumber": "Main Phone Number",
 "ButtonAccountDetails": "ButtonAccountDetails"
 },
 "component": "oj-sp-horizontal-overview",
 "attributes": {
 "id": {
 "$type": "$fn:join",
 "value": [
 "$var:Applet",
 "$var:ButtonAccountDetails"
 ],
 "combiner": ":"
 },
 "itemTitle": "$fn:value($,$var:itemTitle,Value)",
 "displayMode": "light",
 "displayOptions": {
 "showDetails": "on"
 },
 "translations": {
 "$type": "$fn:object",
 "value": {
 "showDetails": "$fn:value($,$var:ButtonAccountDetails,DisplayName)"
 }
 },
 "badge": {
 "$type": "$fn:object",
 "value": {
 "text": "$fn:value($,$var:Status,Value)",
 "status": {
 "$type": "$fn:map",
 "value": "$fn:value($,$var:Status,Value)",
 "mapper": "$fn:execute($,StatusColor)"
 },
 "style": "strong"
 }
 },
```

```
 "onspShowDetails": "$fn:execute($,AccountDetailEvent)"
 }
}
```

This configuration uses dynamic bindings to display record-specific summary information in the overview area.

Configuring the View PM

1. Set the JSON configuration in the custom view PM by using the SEBL_COMPONENT_CONFIG property.

```
this.AddProperty("SEBL_COMPONENT_CONFIG", {
 "imports": [
 "ojs/ojinputsearch",
 "oj-sp/foldout-layout/loader",
 "oj-sp/foldout-panel-summarizing/loader",
 "oj-sp/empty-state/loader",
 "oj-sp/horizontal-overview/loader",
 "ojs/ojlabelvalue",
 "oj-c/input-text",
 "ojs/ojformlayout"
 ],
 "variables": {
 "Applet1": "<Applet Name #1>",
 "Applet2": "<Applet Name #2>",
 "Applet3": "<Applet Name #3>",
 "Applet4": "<Applet Name #4>",
 "Applet5": "<Applet Name #5>",
 "Applet6": "<Applet Name #6>",
 "Applet7": "<Applet Name #7>",
 "AppletTitle": "<Siebel control used for the panel title>",
 "ViewAll": "<Siebel control used for the View All action>",
 "PayBill": "<Siebel control used for a panel action>"
 },
 "component": "oj-sp-foldout-layout",
 "attributes": {
 "orientation": "horizontal",
 "displayMode": "light",
 "animate": "auto",
 "displayOptions": {
 "goToParent": false,
 "inFlowBack": true
 }
 },
 "children": [
 {
 "component": "div",
 "attributes": {
 "slot": "overview"
 },
 "children": [
 {
 "component": "$Applet:($var:Applet1)"
 }
 ]
 }
 ]
});
```

2. Configure the Siebel Manifest for the View.
   a. In Manifest Files, create a new record. Name: siebel/foldoutlayoutmodelviewpm.js
   b. In Manifest Administration, create a new record. Type: View. Usage Type: Presentation Model. Name: <View Name>.
   c. For this record, create a record in Object Expression. Expression: Redwood Theme. Level: 1.

   d. Associate the file siebel/foldoutlayoutmodelviewpm.js with the record by using the View Files list.
3. Configure the Siebel Manifest for the Horizontal Overview Applet.

   a. In Manifest Files, create a new record. Name: siebel/horizontaloverviewwebcomponent.js
   b. In Manifest Administration, create a new record. Type: Applet. Usage Type: Presentation Model. Name: <Applet Name>.
   c. For this record, create a record in the Object Expression applet. Expression: Redwood Theme. Level: 1.
   d. Associate the file siebel/horizontaloverviewwebcomponent.js with the record by using the Applet Files list.
4. Configure the Siebel Manifest for the Foldout Panel List Applet

   The foldout panels can reuse list applets that are configured separately. For example, a panel can render a list applet that uses a custom list-style PM.

   a. In **Manifest Files**, create a new record. Name: siebel/orderlistdashboardappletpm.js
   b. In **Manifest Administration**, create a new record. Type: Applet. Usage Type: Presentation Model. Name: <Applet Name>.
   c. For this record, create a record in the Object Expression applet. Expression: Redwood Theme. Level: 1.
   d. Associate the file siebel/orderlistdashboardappletpm.js with the record by using the Applet Files list.

Validation

- The foldout layout renders correctly in the Redwood theme.
- The overview area displays the expected summary content.
- Each foldout panel displays the expected applet content.
- The panel title and overflow action label display the correct values.
- Overflow actions and secondary actions invoke the expected Siebel behavior.
- Any configured recommendation content is displayed in the appropriate panel slot.
- The reused list applets and overview applets function correctly inside the layout.

Supported Controls in Siebel Web Component Framework

The Web Component Framework supports rendering scenarios through the base Presentation Models and Oracle JET components that are described in the preceding sections of this chapter. Supported controls and behaviors depend on the layout type, the configured base Presentation Model, and the documented JSON bindings that are used for that layout.

The following table lists the supported controls and their corresponding model view implementations.

Control	Model View	Oracle JET Reference
<oj-c-input-text>	`TextModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.InputText.html*

Control	Model View	Oracle JET Reference
<oj-c-button>	`ButtonModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Button.html*
<oj-c-text-area>	`TextAreaModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.TextArea.html*
<oj-combobox-one>	`DropDownModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojComboboxOne.html*
<oj-input-date>	`DateModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojInputDate.html*
<oj-input-date-time>	`DateTimeModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojInputDateTime.html*
<oj-c-labelled-link>	`LinkModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.LabelledLink.html*
<oj-c-checkbox>	`CheckBoxModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Checkbox.html*
<oj-c-radioset>	`RadioModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Radioset.html*
<oj-c-input-password>	`PasswordModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.InputPassword.html*
<oj-c-input-text> and <oj-c-button>	`MailModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.InputText.html%20;%20https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Button.html*
<oj-c-labelled-link>	`URLModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.LabelledLink.html*
<oj-c-input-text> and <oj-c-button>	`MvgPickEDModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.InputText.html%20;%20https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Button.html*
<oj-c-input-date-time> and <oj-c-select-single>	`DateTimeTZModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojInputDateTime.html* *https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.SelectSingle.html*
<oj-chart> <oj-chart-item>	`Chart`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojChart.html* *https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj.ojChartItem.html*
<oj-c-input-text> <oj-c-button> <oj-input-date> <oj-c-form-layout>	`CurrencyModelView`	*https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.InputText.html*

Control	Model View	Oracle JET Reference
		https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.FormLayout.html *https://www.oracle.com/webfolder/technetwork/jet/jsdocs/oj-c.Button.html*

Only the controls, bindings, and behaviors that are documented for the Web Component Framework are supported. If an existing Open UI customization depends on legacy Physical Renderer logic or unsupported control behavior, then additional redesign or customization might be required before that behavior can be used in this framework.

10 Application Programming Interface

Application Programming Interface

This chapter 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*
- *Methods of the Siebel Open UI Application Programming Interface*
- *Methods for Pop-Up Objects and Property Sets*

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*.
- **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*.

For a summary of these methods and information about how Siebel Open UI uses them, see *Life Cycle of User Interface Elements*.

Siebel Open UI defines each class in a separate file. It stores these files in the following folder:

```
AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\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.

For more information about the `language_code`, see *Languages That Siebel Open UI Supports*.

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*
- *Presentation Model Class for Applets*
- *Presentation Model Class for List Applets*
- *Presentation Model Class for Menus*
- *Physical Renderer Class*
- *Plug-in Wrapper Class*
- *Plugin Builder Class*
- *Template Manager Class*
- *Event Helper Class*
- *Business Component Class*
- *Applet Class*
- *Applet Control Class*
- *GetEDEnabled Method*
- *Business Service Class*
- *Application Model Class*
- *Control Builder Class*
- *Locale Object Class*
- *Component Class*
- *Component Manager Class*
- *Other Classes*

Presentation Model Class

This topic describes the methods that Siebel Open UI uses with the PresentationModel class. The methods are described in the following subtopics.

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.

AddLocalString Method

The AddLocalString method adds a text string. It uses the following syntax:

```
AddLocalString(ID, custom_string)
```

where:

- ID is a string that you use to reference the custom_string. You can use any value for ID.
- custom_string is any text string.

For example:

```
this.AddMethod("AddLocalString", function (my_text, this is my custom text) {
SiebelApp.S_App.LocaleObject.AddLocalString(my_text, this is my custom text);
return value;
});
```

This code adds a string named `my_text` that includes the following string value:

```
this is my custom text
```

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*.

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 or a method reference.
- argument and argument_n are arguments that AddMethod sends to the method that methodDefidentifies.
- 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 in the topic *Customizing the Presentation Model to Identify the Records to Delete*:

```
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*.

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",
DerivedPresentationalModel.prototype.MyInvokeControlMethod,{sequence : true});
```

For more information, see *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.

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"));
```

AddValidator Method

The AddValidator method validates an event. It allows you to write a custom validation for any event. It returns one of the following values:

- **true.** Validated the event successfully.
- **false.** Did not validate the event successfully.

It uses the following syntax:

```
Addvalidator(siebConsts.get("event_name"), function(){custom validation}
```

where:

- event_name identifies the name of the event that AddValidator validates.

For example, the following code validates the control focus event:

```
this.AddValidator(siebConsts.get("PHYEVENT_COLUMN_FOCUS"), function(row, ctrl,
val){
if(ctrl.GetDisplayName() === "Account" && val === "Hibbing Mfg"){
return true;
});
```

You can configure Siebel Open UI to use the value that AddValidator returns to determine whether or not to stop running handlers for an event. For more information, see *AttachEventHandler Method*.

For more information about events, see *Siebel CRM Events That You Can Use to Customize Siebel Open UI*.

AttachEventHandler Method

The AttachEventHandler method attaches an event handler to an event. It uses the following values:

- consts.get("SWE_EXTN_CANCEL_ORIG_OP"). If SWE_EXTN_CANCEL_ORIG_OP returns a value of true, then Siebel Open UI cancels the operation for the predefined event handler. For an example that sets the value for SWE_EXTN_CANCEL_ORIG_OP, see *Attaching and Validating Event Handlers in Any Sequence*.

- consts.get("SWE_EXTN_STOP_PROP_OP"). If SWE_EXTN_STOP_PROP_OP returns a value of true, then Siebel Open UI stops the operation for the custom event handler from propagating the customization.

The AttachEventHandler method uses the following syntax:

```
AttachEventHandler(event_name, function_reference);
```

where:

- event_name identifies the name of an event.
- function_reference identifies the name of a method that the AddMethod method adds. For example, PHYEVENT_CONTROL_BLUR. Siebel Open UI calls OnControlEvent to trigger this event, and then calls the function reference in the scope of the corresponding presentation model.

For more information about:

- An example that uses AttachEventHandler, see *Example of the Life Cycle of a User Interface Element*.
- Events, see *Siebel CRM Events That You Can Use to Customize Siebel Open UI*.
- Using AttachEventHandler, see *Life Cycle Flows of User Interface Elements*.
- Deriving a value, see *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.

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*
- Using the AttachNotificationHandle method, see *Customizing Events*
- How Siebel Open UI handles notifications, see *Life Cycle Flows of User Interface Elements*
- Notifications, see *Notifications That Siebel Open UI Supports*

Example of Using AttachEventHandler

Assume a presentation model named pmodel.js includes an OnControlEvent method that runs a custom event handler, and that Siebel Open UI sends an eventConfig object as the last argument in the event handler call. It uses this eventConfig object in the custom presentation model to set a value for SWE_EXTN_CANCEL_ORIG_OP or SWE_EXTN_STOP_PROP_OP. This configuration allows AttachEventHandler to create multiple custom events and to stop an event handler from running.

For example, assume your customization configures Siebel Open UI to do the following:

- Derive derivedpm1.js from pmodel.js.
- Derive derivedpm2.js from derivedpm1.js.
- Derive derivedpm3.js from derivedpm2.js.
- Include an event handler for PHYEVENT_COLUMN_FOCUS in derivedpm1.js, derivedpm2.js, and derivedpm3.js.
- Use derivedpm3.js to set the AttachEventHandler to the value that SWE_EXTN_STOP_PROP_OP contains.
- Use the following code so that Siebel Open UI uses the last argument that AttachEventHandler specifies:

```
this.AttachEventHandler( siebConsts.get("PHYEVENT_COLUMN_FOCUS"), function()
 {
 SiebelJS.Log("Control focus 1");
 arguments[arguments.length - 1][consts.get("SWE_EXTN_STOP_PROP_OP")] = false;
 });
```

Siebel Open UI runs AttachEventHandler customizations in a LIFO (last in, first out) sequence. In this example, it uses the following sequence:

- Runs event handlers that reside in derivedpm3.js.
- Runs event handlers that reside in derivedpm2.js.
- Runs event handlers that reside in derivedpm1.js.
- Runs event handlers that reside in the predefined presentation model.

So, this example stops the custom PHYEVENT_COLUMN_FOCUS event handlers in the derivedpm2.js file and the derivedpm1.js file from running.

How Siebel Open UI Uses AttachEventHandler To Manage an Event

An event occurs when the user clicks an object or changes the focus. To manage this event, Siebel Open UI does the following work:

1. Instructs the physical renderer to call the OnControlEvent method. To make this call, it uses the event name that Siebel Open UI sends to the AttachEventHandler method and corresponding parameters.
2. Identifies the list of event handlers that it registered with the event name in the Init function of the presentation model.
3. Uses the OnControlEvent parameters from Step 1 to call each of the event handlers that it identified in Step 2.
4. Finishes running all the event handlers, and then sends a return value to the object that called OnControlEvent.

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");},{wh
en : 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 Carousel*.

For more information about using the AttachPMBinding method, see *Configuring Siebel Open UI to Bind Methods* and *Life Cycle Flows of User Interface Elements*.

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* and *PostExecute Method*.

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*.

For more examples that use AttachPreProxyExecuteBinding and AttachPostProxyExecuteBinding, see *Customizing the Presentation Model to Call the Siebel Server and Delete a Record* and *Calling Methods for Applets and Business Services*.

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*.

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*.

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*.

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*.

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:

```
undefined
```

It uses the following syntax:

```
this.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* and *Customizing the Presentation Model to Handle Notifications*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*. For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

GetCtrlTemplate Method

The GetCtrlTemplate method gets the template for a control, and then uses values from this template to create an object. It uses values from this template to set the default values and the format for the property set that this control uses. It returns nothing. It uses the following syntax:

```
GetCtrlTemplate ("control_name", "display_name", consts.get( "control_type" ),
column_index);
```

where:

- control_name specifies the name of the control.
- display_name specifies the label that Siebel Open UI displays in the client for this control.
- control_type specifies the type of SWE control, such as SWE_CTRL_TEXTAREA. You can specify one of the following values:
  - SWE_CTRL_URL
  - SWE_CTRL_TEXTAREA
  - SWE_CTRL_TEXT
  - SWE_CTRL_DATE_TZ_PICK
  - SWE_CTRL_DATE_TIME_PICK
  - SWE_CTRL_DATE_PICK
  - SWE_CTRL_CHECKBOX
  - SWE_CTRL_CALC
  - SWE_CTRL_COMBOBOX
  - SWE_CTRL_PWD
- column_index is an integer that specifies the physical location in the list control.

For example, the following code gets the template for the TestEdit control:

```
GetCtrlTemplate ("TestEdit", "Test Edit", consts.get( "SWE_CTRL_TEXTAREA" ), 1);
```

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 in the topic *Overriding Predefined Methods in Presentation Models*.

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. For more information, see *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.

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*.

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 an event.

For more information about:

- Examples that use OnControlEvent, see the following topics:
  - *Modifying CSS Files to Support the Physical Renderer*
  - *Adding Custom User Preferences to Applets*
- How Siebel Open UI uses OnControlEvent, see the following topics:
  - *How Siebel Open UI Uses the Init Method of the Presentation Model*
  - *Siebel CRM Events That You Can Use to Customize Siebel Open UI*
  - *Life Cycle Flows of User Interface Elements*

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*
- *Customizing the Presentation Model to Call the Siebel Server and Delete a Record*
- *Text Copy of Code That Does a Partial Refresh for the Presentation Model*
- *Adding Custom User Preferences to Applets*
- *Using Custom JavaScript Methods*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*

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 more information, see *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.

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*. *Methods That Manipulate Property Sets* and *GetChild Method*.
- Examples that use the Setup method, see *Customizing the Setup Logic of the Presentation Model* and *Adding Presentation Model Properties That Siebel Servers Send for Applets*.
- The Setup method that Siebel Open UI uses with components, see *Setup Method for Components*.

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. The methods are described in the following subtopics.

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*.

Summary of Methods That You Can Use with the Presentation Model for Applets

The following table 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.

Method	Callable	Bindable
CanInvokeMethod	Yes	No
CanNavigate	Yes	No
CanUpdate	Yes	No
ExecuteMethod	Yes	No
ExecuteUIUpdate	No	Yes
FieldChange	No	Yes

Method	Callable	Bindable
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
SetUpdateConditionals	Yes	No
ShowPickPopup	Yes	No

Method	Callable	Bindable
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

The following table lists the 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 the Carousel*.
GetMode	Returns a string that identifies the applet mode.
GetName	Returns a string that includes the name of the presentation model.

Property	Description
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 Siebel Open UI uses it in query mode.
GetRecordset	Returns an array that lists the record set that the applet currently displays.
GetSelection	Returns the number of records the user chooses in the applet.
GetTitle	Returns a string that includes the applet title that the presentation model defines.

GetUIEventMap	Returns an array that lists user interface events that are pending. Each element in this array identifies an event that you can access using the following code:
	`this.Get("GetUIEventMap") [ index ].ev`
	You can use the following code to access the arguments:
	`as this.Get("GetUIEventMap") [ index ].ar`

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:

```
CanInvokeMethod(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*.

For more examples that use CanInvokeMethod, see the following topics:

- *Customizing the Presentation Model to Delete Records*
- *Attaching an Event Handler to a Presentation Model*
- *Customizing Applets to Capture Signatures from Desktop Applications*
- *Customizing a Resource Scheduler*
- *Using Custom JavaScript Methods*
- *Using Custom Siebel Business Services*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Commit Part Tracker Records*

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.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.GetPM().Get("GetControls");
for(var control in controlSet){
 if(controlSet.hasOwnProperty(control)){
 var canNavigate = this.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.GetPM().Get("GetControls");
for(var control in controlSet){
 if(controlSet.hasOwnProperty(control)){
 var canupdate = this.GetPM().ExecuteMethod("CanUpdate", controlSet[ control
].GetName());
 }
}
```

For an example that uses the CanUpdate method, see *UpdateRecord Method*.

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*.
- *Customizing the Presentation Model to Delete Records*
- *Customizing the Presentation Model to Handle Notifications*
- *Calling Methods*
- *Accessing Proxy Objects*

For information about how Siebel Open UI uses the ExecuteMethod method, see *How Siebel Open UI Uses the Init Method of the Presentation Model*.

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*.

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*
- An example that uses the FieldChange method, *Displaying and Hiding Fields*
- Using this method, see *Life Cycle Flows of User Interface Elements*
- The FieldChange method that Siebel Open UI uses with physical renderers, see *FieldChange Method for Presentation Models*

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*.

For information about where you add this code, see *Adding Code to the Physical Renderer*.

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*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*

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 ToggleEl = this.GetPM().ExecuteMethod("GetControlId");
```

For information about where you add this code, see *Adding Code to the Physical Renderer*.

GetFieldValue_Method

The GetFieldValue method returns the value of a field. It uses the following syntax:

```
this.GetFieldValue(field_ame);
```

where:

- field_name identifies the name of a field.

For example, the following code gets the current value of the Call Status field:

```
pBusComp.GetFieldValue("Call Status");
```

For another example that uses the GetFieldValue method, see *Text Copy of Code That Does a Partial Refresh for the Presentation Model*.

GetFormattedFieldValue Method

The GetFormattedFieldValue method gets the format that a field uses to store the value of a control. It uses the following syntax:

```
value = this.GetPM().ExecuteMethod("GetFormattedFieldValue", 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*.

You add the GetFormattedFieldValue method to the physical renderer.

Siebel Open UI gets the format 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*.

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*.

For information about where you add this code, see *Adding Code to the Physical Renderer*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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* and *Customizing Methods in the Presentation Model to Store Field Values*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

For more information about using this method, see *AttachPostProxyExecuteBinding Method* and *Life Cycle Flows of User Interface Elements*.

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*.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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

For examples that use the SetActiveControl method, see the following topics:

- *Customizing the Presentation Model to Identify the Records to Delete*
- *Customizing the Presentation Model to Delete Records*
- *Accessing Proxy Objects*
- *AddMethod Method*

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(isActive, newActiveApplet)
```

For example:

```
this.SetHighlightState(isActive);
```

You can add the following code to the physical renderer:

```
this.AttachPMBinding("SetHighlightState", function(isActive, 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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

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* and *Example of Adding a New Method*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

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*.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements* and *Notifications That Siebel Open UI Supports*.

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. The methods are described in the following subtopics.

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*

Properties of the Presentation Model That Siebel Open UI Uses for List

The following table lists the properties of the presentation model that Siebel Open UI uses for a list applet.

Property	Description
GetBeginRow	Returns the beginning row number of a list applet.
GetListOfColumns	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: • name • controlType • isLink • index • bCanUpdate • control For more information about JSON, see the JSON website at: `http://www.json.org`.
GetRowIdentifier	Returns a string that contains information about the row.
GetRowListRowCount	Returns the number of rows that a list applet contains.
GetRowsSelectedArray	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: • **true.** The row is chosen.

Property	Description
	• **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

The following table 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.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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);
```

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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*.

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*.

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*:

```
this.ExecuteMethod("OnDrillDown", my_accounts, counter);
```

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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.
  - **pvrc.** Scroll up to the previous record.
  - **nxst.** Page down to the next set of records.
  - **pvst.** Page up to the prior set of records.
  - **scrolldn.** Scroll down one page.
  - **scrollup.** Scroll up one page.

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");
```

SetMultiSelectMode Method

The SetMultiSelectMode method determines whether or not a list applet is using multiselect mode. It uses the following syntax:

```
SetMultiSelectMode(bInMultiSelMode)
```

where:

- `bIsInMultiSelectMode` 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(bInMultiSelMode);
```

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(bIsInMultiSelectMode){
 }
```

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. The methods are described in the following subtopics.

Properties of the Presentation Model for Menus

The following table describes the properties of the presentation model that Siebel Open UI uses 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:

```
childArray
[0]
- childArray
```

```
- 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:

```
activeAplt.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. The methods are described in the following subtopics.

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*
  - *Siebel Portal Framework*
- How Siebel Open UI uses BindData, see the following topics:
  - *Life Cycle of a Physical Renderer*
  - *Guidelines for Customizing Presentation Models*

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* and *Siebel CRM Events That You Can Use to Customize Siebel Open UI*.

For information about how Siebel Open UI uses BindEvents, see the following topics:

- *Life Cycle of a Physical Renderer*
- *Guidelines for Customizing Presentation Models*
- *Life Cycle Flows of User Interface Elements*

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*
- *Guidelines for Customizing Presentation Models*
- *Using Methods with the Base Physical Renderer Class*

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*.

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*.

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 the Carousel*
  - *Customizing the Physical Renderer to Bind Events*
  - *Customizing the Physical Renderer to Refresh the Carousel*
  - *Calling Methods for Applets and Business Services*
  - *Refreshing Custom Events*
  - *Displaying and Hiding Fields*
  - *Adding Custom User Preferences to Applets*
- How Siebel Open UI uses GetPM, see the following topics:
  - *Life Cycle of a Physical Renderer*
  - *Using Methods with the Base Physical Renderer Class*
- The GetPM method that Siebel Open UI uses for components, see *GetPM Method for Components*.

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*.

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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 the Carousel*
  - *Adding Custom User Preferences to Applets*
- How Siebel Open UI uses ShowUI, see the following topics:
  - *Life Cycle of a Physical Renderer*
  - *Using Methods with the Base Physical Renderer Class*
  - *BindEvents Method*

Plug-in Wrapper Class

This topic describes the methods that Siebel Open UI uses with the basepw, which is the Plug-in Wrapper base class. The methods are described in the following subtopics.

GetEl Method

The GetEl method simplifies the process of finding DOM element associated with a particular control in the applet region. It can detect if the control has multiple instances in the DOM and if so, it will them return all. If a single instance is required, the index must be passed to this function. It uses the following syntax:

```
GetEI(index)
```

- Where index is a numerical value representing the row number of the DOM element of the control that is required. This argument is optional.

Returns the associated jQuery based DOM reference for the control or NULL.

For example, the following code uses the GetUI method to retrieve all DOM element of a particular control:

```
var el = this.GetUIWrapper( control ).GetEl();
```

For another example, the following code uses the GetUI method to retrieve index-based DOM elements of a particular control when the control has multiple DOM instances, as in a list applet:

```
var el = this.GetUIWrapper( control ).GetEl( index );
```

ShowUI_Method

The ShowUI method performs show-related activities for a control. It requires the GetEl method and the Template Manager to accomplish its purpose.

BindEvents_Method

The BindEvents method attaches events to the DOM instance of a control. It requires the GetEl method and the Event Helper to accomplish its purpose.

For more information, see *BindEvents Method*.

SetValue Method

The SetValue method sets the value in the DOM instance of control. If there are multiple DOM instances for the control, the index argument is used to used to determine the instance to which the value should be set.Customized plug-in wrappers must use this index to find associated DOM instances and call appropriate value modification APIs in the DOM to reflect the customization.

It uses the following syntax:

```
SetValue(value, index)
```

where:

- value identifies the value of the control DOM instance.
- index is a numerical value representing the row number of the DOM element of the control that is required.

GetValue Method

The GetValue method gets the value of the control field from the DOM. If multiple instances of the control exist, then the index parameter is used to identify the value of the particular control that is needed. It uses the following syntax:

```
GetValue(index)
```

- Where index is a numerical value representing the row number of the DOM element of the control that is required.

BeginQuery Method

The BeginQuery method indicates to a customized PW that it is entering query mode. It uses the following syntax:

```
BeginQuery()
```

It includes no arguments.

EndQuery Method

The EndQuery method indicates to a customized PW that it is exiting query mode. It uses the following syntax:

```
EndQuery()
```

It includes no arguments.

GetIconMap Method

The GetIconMap method determines if there are any configured icon maps for a customized PW control. If it does, the appropriate icon map is returned. It uses the following syntax:

```
GetIconMap()
```

It includes no arguments.

SetState Method

The SetState method provides an indicator to a customized PW to set a state to the associated DOM instances. If there are multiple DOM instances, use the index argument to retrieve the appropriate element. It uses the following syntax:

```
SetState(state, flag, index)
```

where:

- state is one of the following values:
  - **EDITABLE.** Can be edited.
  - **ENABLE.** Is enabled.
  - **SHOW.** Is visible.
  - **FOCUS.** Is focussed.
- flag indicates if the state should be reversed, and is one of the following values:
  - **TRUE.** The state should be reversed.
  - **FALSE.** The state should be maintained.

    For example, if the state is set to EDITABLE, and the flag is set to TRUE, the value of state will be reversed to NON-EDITABLE.
- index is a numerical value representing the row number of the DOM element of the control that is required.

Plugin Builder Class

This topic describes the Plugin Builder class. The Plugin Builder class wires the Plug-in Wrapper to the given Applet Control, specifying the conditions under which the wrapper is to be used. It uses the API AttachPW for this purpose. It uses the following syntax:

```
SiebelApp.S_App.PluginBuilder.AttachPW(Control Type, PW Class, function (control,
objName) {
 return <conditions>;
```

where:

- Control Type is the SWE constant for the type of control that you are trying to override the functionality for. For a complete listing of control types, see *About Supported Template Manager Controls*.
- PW Class is the name of the custom wrapper.

For example, the following code shows how to attach the Plug-In wrapper with a custom combobox wrapper that would deploy for all buttons in the Contact List Applet:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_COMBOBOX"),
SiebelAppFacade. CustomComboPW, function (control, objName) {
 return (objName === "Contact List Applet");
});
```

Another example, the following code shows how to attach the Plug-In wrapper with a custom text box wrapper that would deploy for all text boxes in the Opportunity List Applet or the Sales Order Form Applet:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_TEXT"),
SiebelAppFacade.CustomTextPW, function (control, objName) {
return (objName === "Opportunity List Applet" || objName === "Sales Order Form");
});
```

Another example, the following code shows how to attach the Plug-In wrapper with a custom combobox wrapper that would deploy for all combo boxes of a certain name, across the application:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_COMBOBOX"),
SiebelAppFacade.CustomComboPW, function (control, objName) {
return (control.GetName() === "Last Name");
});
```

Another example, the following code shows how to attach the Plug-In wrapper with a custom text box wrapper that would deploy for only a specific control with a specific name in the Sales Order Form Applet:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_TEXT"),
SiebelAppFacade.CustomTextPW, function (control, objName) {
 return (control.GetName() === "Revenue" && objName === "Sales Order Form");
});
```

Another example, the following code shows how to attach the Plug-In wrapper with a custom check box that would deploy for all touch enabled devices:

```
SiebelApp.S_App.PluginBuilder.AttachPW(consts.get("SWE_CTRL_CHECKBOX"),
SiebelAppFacade.CustomCheckPW, function (control) {
return SiebelAppFacade.DecisionManager.IsTouch();
});
```

Note: The global call depicted in this example can be used in conjunction with other conditions, such as the ones in previous examples.

For more information about the Attach PW API and examples of how to use the AttachPW API, see *Configuring the Manifest for the Color Box Example*.

Template Manager Class

This topic describes the Template Manager Class. The methods are described in the following subtopics.

About the Template Manager Class

The Template Manager class generates HTML for various controls, and uses the following method and syntax:

```
GenerateMarkup( configObject );
```

The GenerateMarkup method uses only one argument, that is an object. Depending on the properties present in object, Template Manager chooses the appropriate flow for the generation of the HTML. The following list describes the different properties that you can specify via configObject:

- **type.** Specify the type of control to generate. For a list of types, please see the following table. When not specified, the value will default to the input field or SWE_CTRL_TEXT.
- **class.** Specify the class name to attach to the control. If multiple classes need to be attached, use a space-separated string. TM will also attaches pre-defined CSS class name for the control, based on the type of control being generated.
- **id.** Specify the ID which needs to be given to the control. Depending on the control type provided, auto generated value for ID can be attached by TM to the control if not provided.
- **values.** Specify the value that needs to be attached to the control.

  > **Note:** When specifying ComboBox for the type, you can specify an array of values. Also, the selected index needs to be specified with the property index.

- **attrs.** Specify any other attribute that should be attached to the control, in string format. For example, if you need aria attributes `aria-label`, `aria-labelledby`, and `aria-describedby` to be attached to the control, you would use the following code:

  ```
  "aria-label='abc' aria-labelledby='xyz' aria-describedby='123'"
  ```

About Supported Template Manager Controls

This topic describes supported Template Manager controls.

The Template Manager class provides mark-up for the many types of controls required in Siebel Open UI. The following table lists the supported Template Manager controls.

HTML Type in Siebel Open UI	Corresponding SWE Constants	Additional Information
Button	SWE_PST_BUTTON_CTRL	None.
Text Field	SWE_CTRL_TEXT	None.
span	SWE_CTRL_PLAINTEXT	None.
Check Box	SWE_CTRL_CHECKBOX	None.
Date (HTML5)	SWE_CTRL_DATE_PICK	Falls back to HTML4 input field control.

HTML Type in Siebel Open UI	Corresponding SWE Constants	Additional Information
Date Time (HTML5)	SWE_CTRL_DATE_TIME_PICK	Falls back to HTML4 input field control.
URL (HTML5)	SWE_CTRL_URL	None.
TEL (HTML5)	SWE_CTRL_PHONE	Falls back to HTML4 input field control.
File	SWE_CTRL_FILE	None.
Radio	SWE_CTRL_RADIO	None.
Eff Date	SWE_CTRL_EFFDAT	None.
MVG	SWE_CTRL_MVG	None.
Pick	SWE_CTRL_PICK	None.
Detail	SWE_CTRL_DETAIL	None.
Calc	SWE_CTRL_CALC	None.
Link	SWE_CTRL_LINK	Links with the address in src property.
MailTo	SWE_CTRL_MAILTO	Links with the address supplied in src property.
Img	SWE_CTRL_IMAGECONTROL	Image with the source provided in src property.
Text Area	SWE_CTRL_TEXTAREA	None.
Label	SWE_CTRL_LABEL	None.
ComboBox (Select)	SWE_CTRL_COMBOBOX	Accepts the following additional configuration: • **displayValue.** An array of values that should be displayed in the Option List. • **value.** An array of actual values.

HTML Type in Siebel Open UI	Corresponding SWE Constants	Additional Information
		• **index.** Zero-based value that indicates currently selected value.

Examples Using Template Manager

This topic describes examples of using Template Manager. The examples in this section assume that `tmplMgr` is a reference to the Template Manager Object, and `consts` is a reference to SiebelApp.Constants object.

Example of Generating Markup for a Normal Text Field

In this example, we use the following code make the call to the Template Manager to generate markup for a normal text field:

```
var markup = tmplMgr.GenerateMarkup({
 type : consts.get( "SWE_CTRL_TEXT" )
 });
```

In the this example, this is the expected HTML string begin held by the markup variable:

```
<input type="text" class="siebui-input " />
```

Example of Generating Markup with an Additional className

In this example, we use the following code make the call to the Template Manager to generate markup for with an additional className:

```
var markup = tmplMgr.GenerateMarkup({
 type : consts.get( "SWE_CTRL_TEXT" ),
 class: "siebui-align-left"
 });
```

In the this example, this is the expected HTML string begin held by the markup variable:

```
<input type="text" class="siebui-input siebui-align-left" />
```

Example of Generating Markup with Additional Attributes

In this example, we use the following code make the call to the Template Manager to generate markup for with additional attributes:

```
var markup = tmplMgr.GenerateMarkup({
 type : consts.get( "SWE_CTRL_TEXT" ),
 attrs: "aria-label=\"abc\" aria-labelledby=\"xyz\" aria-describedby=\"123"
 });
```

In the this example, this is the expected HTML string begin held by the markup variable:

```
<input type="text" class="siebui-input " aria-label="abc" aria-labelledby="xyz"
aria-describedby="123 />
```

Example of Generating a Combo Box with Multiple Options

In this example, we use the following code make the call to the Template Manager to generate a combo box with multiple options Value 1, Value 2, and Value 3:

```
var markup = tmplMgr.GenerateMarkup({
 type : consts.get( "SWE_CTRL_COMBOBOX" ),
```

```
 value: [ "Value 1", "Value 2", "Value 3" ],
 index: 1 // zero based index
 });
```

In the this example, this is the expected HTML string begin held by the markup variable:

```
<select class="siebui-select ">
 <option>Value 1</option>
 <option selected> Value 2</option>
 <option>Value 3</option>
</select>
```

Example of Generating a Hyperlink

In this example, we use the following code make the call to the Template Manager to generate a hyperlink:

```
var markup = tmplMgr.GenerateMarkup({
 type : consts.get( "SWE_CTRL_LINK" ),
 src : "http://www.oracle.com",
 value: "Oracle HomePage"
 });
```

In the this example, this is the expected HTML string begin held by the markup variable:

```
<a class="siebui-link" src="http://www.oracle.com>Oracle HomePage</a>
```

Event Helper Class

The Event Helper Class uses the Event Helper Object to facilitate Event Binding in the Physical Renderer or Plug-in Wrapper.

To retrieve the event helper:

```
var evtHelper = this.Helper("EventHelper" );
```

Manage is the singular API exposed by the Event Helper Class for unified event binding for DOM elements across multiple platforms. Use the following API specification as a guideline to use the EventHelper object to bind an event:

```
evtHelper.Manage( el, eventName, eventData, eventHandler );
```

> **Note:** This syntax is similar to a jQuery bind call. With this call, an attempt is being made to bind event `eventName` to element `el` with event data `eventData` and event handler `eventHandler`.

Use the following API specification as a guideline to use delegate-on type for event binding:

```
evtHelper.Manage( el, eventName, eventData, eventHandler, elChild );
```

For example:

```
var evtHelper = this.Helper( "EventHelper" );
evtHelper.Manage( el, "down" , functionRef );
```

The `down` event is attached to element `el`, with `functionRef` defined as the Event Handler. Both touch and mouse events are handled, depending on the environment. The `down` value will get translated to `mousedown` in a mouse-enabled environment, and to `touchstart` in a touch-enabled environment.

About Event Helper Mappings

Inter-platform event mappings done by the Event Helper object are harmonized, in that similar actions that create different events on different platforms result in the same behavior across platforms.

The following table shows unified event names and their corresponding actions on touch and non-touch platforms. Using the new unified events creates familiar experiences for users across platforms.

Unified Event Name	Translation On Non-Touch Platform	Translation On Touch Platform
down	mousedown	touchstart
start	mousedown	touchstart
click	click	click
up	mouseup	touchend
end	mouseup	touchend
move	mousemove	touchmove
over	mouseover	none
out	mouseout	none
cancel	mouseout	touchcancel
dnter	mouseenter	none
leave	mouseleave	none
hover	hover	none
focus	focus	focus
blur	blur	blur
keydown	keydown	none
keyup	keyup	none
keypress	keypress	keypress

Furthermore, the same unified bindings translate to pointer-based events if the Siebel Open UI Client application detects that the browser supports the pointer object. This behavior is specific to Internet Explorer browsers and pointer events used by Microsoft to unify event handling across different devices on Internet Explorer 10 and later.

The following table describes the pointer event mapping.

Unified Event Name	Translation On Windows 8 Internet Explorer Pointer-Based Devices
down	pointerdown
start	pointerdown
click	click
up	pointerup
end	pointerup
move	pointermove
over	pointerover
out	pointerout
cancel	pointercancel
enter	pointerenter
leave	pointerleave
hover	mspointerhover
focus	focus
blur	blur
keydown	keydown
keyup	keyup
keypress	keypress

About Double-Click

A double click event is usually handled natively by the browser, such as the zoom action in touch based devices. Consequently, it is not recommended that you attach custom handlers to the double-click event. Attaching custom handlers might make it impossible to unify the behavior of the double-click action.

About Events Not Unified by Event Helper

Events not unified by the Event Helper or listed in the tables in the topic *About Event Helper Mappings* can still be used with Manage API to attach custom handlers. This applies to events supported by jQuery natively and to custom events that are generated by custom PR/PW code or by third-party plug-in customizations.

For example, a plug-in like iScroll might trigger events such as `scollLeft` or `scrollStop` on the element to which the plug-in is attached. The custom PR code can effectively attach custom handlers to these events using the Manage API.

API Specification for Context Renderers

This topic describes the methods that Siebel Open UI uses with Context Renderer base class. Any customized context renderer must extend from Base Context Renderer (SiebelAppFacade.BaseCR) and implement following functions:

- **Init Method.** Siebel Open UI framework calls the Context Renderer's Init function after instantiating the Context Renderer object. This method is called before the execution of Physical Renderer's life cycle events. For information about physical renderer life cycle events, see *Life Cycle of a Physical Renderer*.

  Siebel Open UI also injects the corresponding physical renderer's instance to this function. The implementation calls the methods supported via the physical renderer's instance.

  For example, if the customized context renderer needs to attach a method binding for ShowSelection, it can call AttachPMBinding by implementing the Init function as follows (this example assumes that the method OnShowSelection is defined in the customized Context Renderer):

  ```
  CustomCR.prototype.Init = function ( prContext ) {
   prContext.AttachPMBinding ( "ShowSelection", OnShowSelection);
   }
  ```

  prContext refers to the corresponding physical renderer instance injected by the Siebel Open UI framework while calling this function.

- **Execute Method.** Siebel Open UI framework calls the Context Renderer's Execute function immediately after the execution of Physical Renderer's life cycle events. For information about physical renderer life cycle events, see *Life Cycle of a Physical Renderer*.

  Siebel Open UI also injects the corresponding physical renderer's instance to this function. The implementation calls the method supported via the Physical Renderer's instance.

  For example, if the context renderer needs to determine the available recordset, it can implement the function as follows:

  ```
  CustomCR.prototype.Execute = function ( prContext ) {
   var recordSet = prContext.GetPM().Get( "GetRecordSet" );
   }
  ```

  prContext refers to the corresponding physical renderer instance injected by the Siebel Open UI framework while calling this function.

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*.

Applet Class

This topic describes the methods that Siebel Open UI uses with the Applet class. The methods are described in the following subtopics.

Siebel Open UI defines this class in the applet.js file.

AddClientControl Method

The AddClientControl method adds a control in the client. It returns nothing. It uses the following syntax:

```
Applet.prototype.AddClientControl = function (ctrlInfo) {
....
}
```

It includes no arguments.

For an example that uses the GetControls method, see *Customizing Methods in the Presentation Model to Store Field Values*.

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:

```
GetControls()
```

It includes no arguments.

For an example that uses the AddClientControl method, see *Creating and Managing Client-Side Controls*.

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* see *GetName Method for Application Models*.

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:

```
GetSelection()
```

It includes no arguments.

Applet Control Class

This topic describes the methods that Siebel Open UI uses with the Applet Control class. The methods are described in the following subtopics.

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 the 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 the 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* and *CanNavigate Method*.

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*
- *GetPopupHeight Method*
- *GetPrompt Method*

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*.

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*.

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*
- *Overriding Predefined Methods in Presentation Models*
- *Text Copy of Code That Does a Partial Refresh for the Presentation Model*
- *CanNavigate Method*
- *ExecuteMethod Method*
- *IsPrivateField Method*
- *CellChange Method*

For information about the GetName method that Siebel Open UI uses for other classes, see *GetName Method for Applets* see *GetName Method for Application Models*.

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*.

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*.

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*.

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.

HandleDeleteNotification Method

The HandleDeleteNotification method deletes the reference to record data that Siebel Open UI stored in the client for a control. For an example that uses the HandleDeleteNotification method, see *Creating and Managing Client-Side Controls*.

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:

```
IsEditEnabled()
```

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 (bInsertBefore, bInternal, pIdValue) {}
```

where:

- `bInsertBefore` 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.
- `bInternal` 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.
- `pIdValue` contains the value that Siebel Open UI uses as the Id for the new record. You can specify a value for pIdValue to create a new record with a row Id that you specify. If you do not specify pIdValue, 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*

- *Calling Methods*
- *Allowing Users to Return Parts*
- *SetMultipleFieldValues Method*
- *UpdateRecord Method*
- *AttachPostProxyExecuteBinding Method*

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* and *SetErrorMsg Method*.
- 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.

PreGetFormattedFieldValue Method

The PreGetFormattedFieldValue method gets the format that a field uses to store the value of a control. For an example that uses the PreGetFormattedFieldValue method, see *Creating and Managing Client-Side Controls*.

PostLeaveField Method

The PostLeaveField method temporarily stores a value that the user enters in a control. It stores this value in memory. You use the AddMethod to call the PostLeaveField method. Siebel Open UI then implicitly calls the PostLeaveField method from the LeaveField method that the listapplet.js file specifies. For an example that uses the PostLeaveField method, see *Creating and Managing Client-Side Controls*.

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 indeces 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. The methods are described in the following subtopics.

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*.

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:

- rowld 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*.

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_TASKUISVC"));
```

For more information, see *GetService Method*.

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. The methods are described in the following subtopics.

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*.

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.example.com/siebel/app/callcenter/enu?
SWECmd=LogOff&SWEKeepContext=1&SWERPC=1&SRN=L8ct6oeEsPA3Cj7pF6spebyCLm2m
VGpB0D0tqGMcflcb&SWEC=18&SWEActiveApplet=Client Active Applet&SWEActiveView=Client
Active View
```

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 = SiebelApp.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*
- *Customizing Browser Tab Labels*
- *Displaying Data from External Applications in Siebel Open UI*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Return Parts*
- *Allowing Users to Set the Activity Status*
- *Name Method for Applets*

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 = SiebelApp.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*.

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:

```
GetAppTitle()
```

It includes no arguments.

For example:

```
var appTitle = SiebelApp.S_App.GetAppTitle();
if (appTitle === "Siebel Call 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 progressing forward from first-to-last.
- **Null.** Siebel Open UI is not configured so the user reads text progressing backwards from last-to-first.

It uses the following syntax:

```
GetDirection()
```

It includes no arguments.

GetName Method for Application Models

The GetName method that Siebel Open UI uses for application models. It returns the name of the current Siebel application, such as Siebel Call Center. It returns this name in a string. It uses the following syntax:

```
GetName()
```

It includes no arguments.

For example:

```
activeView.ExecuteFrame (activeApplet.GetName(), [{field: this.Get("SearchField"),
value: this.Get("SearchValue")}])
```

For information about the GetName method that Siebel Open UI uses for other classes, see *GetName Method for Applets* see *GetName Method for Applet Controls*.

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.example.com/siebel/start.swe
```

It uses the following syntax:

```
GetPageURL()
```

It includes no arguments.

For example:

```
finalurl = SiebelApp.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.

Attributes supported are:

- **OperationalMode.** The mode of the applet as configured in Applet Web Template. The returned value can be one of the following: Base, Edit, EditList, New, or Query.
- **VisualMode.** The applet visualization, which 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.

For examples that use the GetProfileAttr method, see *Adding Custom Manifest Expressions* and *Configuring Siebel Open UI to Use Different Web Templates According to the Applet Mode*.

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:

```
SiebelApp.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 = SiebelApp.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 Applets and Business Services*
- *RemoveService Method*

For information about Siebel Open UI uses GetService with InvokeMethod, see *Invoke Method for Business Services*.

GotoView Method

The GotoView method navigates the user to a view in the client. It uses the following syntax:

```
SiebelApp.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:

```
SiebelApp.S_App.GotoView(view, viewId, strURL, strTarget);
```

For more examples that use the GotoView method, see the following topics:

- *SetDiscardUserState Method*
- *Displaying Siebel Portlets in External Applications*
- *Using iFrame Gadgets to Display Siebel CRM Applets in External Applications*

For more information about using this method, see *Life Cycle Flows of User Interface Elements*.

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:

```
SiebelApp.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 example, the following code calls the NextApplet method. This method sets the next applet as the active applet of a view:

SiebelApp.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*
- *Attaching an Event Handler to a Presentation Model*
- *Calling Methods for Applets and Business Services*
- *Using Siebel Business Services or JavaScript Services to Customize Siebel CRM Objects*
- *Using Custom JavaScript Methods*
- *Using Custom Siebel Business Services*
- *Customizing Siebel Pharma for Siebel Mobile Disconnected Clients*
- *Allowing Users to Commit Part Tracker Records*
- *Allowing Users to Return Parts*

For more information about using InvokeMethod, see *Calling Methods for Applets and Business Services*.

For more information about the InvokeMethod method that Siebel Open UI uses for other classes, see *InvokeMethod Method for Presentation Models* and *InvokeMethod Method for Business Services*.

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.

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*.

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*.

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.
SiebelApp.S_App.DiscardUserState (true);
// Don't care about user state - we need the navigation to occur.
SiebelApp.S_App.GotoView ("MyView"..);
// Reset
SiebelApp.S_App.DiscardUserState (false);
```

Control Builder Class

The following table describes the methods that Siebel Open UI uses with the ControlBuilder class.

Method	Description
Pick Mvg	You can use the following properties of the configuration object: • **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. • **className.** Modifies the CSS style of the pick icon.
DatePick DateTimePick	You can use the following properties of the configuration object: • **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. DateTimePick does the same as DatePick except the dialog box allows the user to set the date and time. • **className.** Identifies the class.

Locale Object Class

This topic describes the methods that Siebel Open UI uses with the Locale Object class. The methods are described in the following subtopics.

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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.GetDispCurrencyDecimal()
```

GetDispCurrencySeparator Method

The GetDispCurrencySeparator method 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.LocaleObject.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.LocaleObject.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.LocaleObject.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.LocaleObject.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:

```
GetDispNumberSeparator()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocaleObject.GetDispNumberSeparator())
```

GetDispTimeAM Method

The GetDispTimeAM method returns the localized string for AM. For example, AM. It uses the following syntax:

```
GetDispTimeAM()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocaleObject.GetDispTimeAM()
```

GetDispTimePM Method

The GetDispTimePM method returns the localized string for PM. For example, PM. It uses the following syntax:

```
GetDispTimePM()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocaleObject.GetDispTimePM()
```

GetDispTimeSeparator Method

The GetDispTimeSeparator method returns the symbol that the client uses to separate the parts of time. For example, the colon (:) symbol. It uses the following syntax:

```
GetDispTimeSeparator()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocaleObject.GetDispTimeSeparator()
```

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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.GetFuncCurrCode()
```

GetLocalString Method

The GetLocalString method returns the localized string of a key. It uses the following syntax:

```
GetLocalString(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 GetLocalString gets.

For example, the following code uses the GetLocalString method when using Siebel Open UI with a connected client:

```
SiebelApp.S_App.LocaleObject.GetLocalString("IDS_SWE_LOADING_INDICATOR_TITLE")
```

For another example, the following code uses the GetLocalString method when using Siebel Open UI with Siebel Mobile Disconnected:

```
SiebelApp.S_App.OfflineLocaleObject.GetLocalString("IDS_SWE_LOADING_INDICATOR_TITL
E")
```

GetMonth Method

The GetMonth method returns the month that the locale uses. It uses the following syntax:

```
GetMonth()
```

It includes no arguments.

For example:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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:

```
SiebelApp.S_App.LocaleObject.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*.

For example:

```
SiebelApp.S_App.LocaleObject.StringToFormatted("date","11/05/2012","M/D/YYYY")
```

Component Class

This topic describes the methods that Siebel Open UI uses with the Component class. The methods are described in the following subtopics.

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 SiebelAppFacade.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*.

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*.

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*.
- 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.

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*.

For information about the Show method that Siebel Open UI uses for component managers, see *Show Method for Component Managers*.

Component Manager Class

This topic describes the methods that Siebel Open UI uses with the Component Manager class. The methods are described in the following subtopics.

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*.
- 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*.

For an example that uses the MakeComponent method, see *Creating Components*.

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 *Setup Method for Components*.

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*.
- 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("SiebelAppFacade.module_name") === undefined){
 SiebelJS.Namespace("SiebelAppFacade.module_name");
define("siebel/custom/module_name", [], function(){
 SiebelAppFacade.module_name = (function(){
 var consts = SiebelJS.Dependency("SiebelApp.Constants");
 function module_name (){
 SiebelAppFacade.module_name.superclass.constructor.apply(this,
arguments);;
 };
 SiebelJS.Extend(module_name, SiebelAppFacade.arguments_2);
 return module_name:
 })();
 return SiebelAppFacade.module_name;
 });
}
```

where:

- SiebelAppFacade 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:

  ```
  RecycleBinPModel
  ```

- `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 the figure under the topic *Creating the Presentation Model*.
- A physical renderer, see the figure under the topic *Setting Up the Physical Renderer*.

For information about how to add manifest files and manifest expressions that reference the module_name, see *Configuring Manifests*.

ShowErrorMessage Method

The ShowErrorMessage method specifies the error message that Siebel Open UI displays. It returns nothing. It uses the following syntax:

```
ShowErrorMessage(error_message)
```

where:

- *error_message* is a string that contains the text of the error message.

Methods for Pop-Up Objects and Property Sets

This topic describes the methods that Siebel Open UI uses with pop-up objects and property sets. It includes the following information:

- *Pop-Up Presentation Models and Physical Renderers*
- *Methods That Manipulate Property Sets*

Pop-Up Presentation Models and Physical Renderers

The PopupPModel presentation model specifies how to model pop-up objects. It uses the following syntax:

```
SiebelApp.PopupPModel
```

The PopupRenderer physical renderer specifies how to render pop-up objects. It uses the following syntax:

```
SiebelAppFacade.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:
  a. Gets the layout and data for the pop-up object.

b. 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

The following table 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.

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: • **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.
height	Specifies the height of the pop-up object, in pixels.
isCancelQryPopupOpen	Includes one of the following return values: • **true.** A cancel query object is open. • **false.** No cancel query objects are open.
isCurrencyOpen	Includes one of the following return values: • **true.** A currency pop-up object is open. • **false.** No currency pop-up objects are open.
isPopupClosedByX	Includes one of the following return values: • **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.

Property	Description
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: • **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: • **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 • **open.** Siebel Open UI opens an empty dialog box. • **close.** Siebel Open UI closes an open dialog box.
url	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: • **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. 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.
width	Specifies the width of the pop-up object, in pixels.

Methods of the Popup Presentation Model

The following table 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.

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.

Method Name	Description
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.
ProcessNewPopup(propset)	Processes the property set that Siebel Open UI sends to this method as an argument, and then determines the following items: • The mode that the pop-up object uses • Various pop-up window features • The width and height of the pop-up object, in pixels. 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.
SetPopupVisible(bVisible)	Modifies the state property of the pop-up object depending on whether the bVisible argument is true or false.

Methods of the Popup Physical Renderer

The following table describes the methods of the PopupRenderer physical renderer.

Method Name	Description
BindEvents	Binds all events for the pop-up object. For more information, see *Siebel CRM Events That You Can Use to Customize Siebel Open UI*.
EnhanceDialog	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. Siebel Open UI calls the EnhanceDialog method when it calls the OnLoadPopupContent method from the PopupPM presentation model.
LoadContent	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.
LoadURL	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.
SetTitle	Sets the title for the pop-up object. Siebel Open UI calls the SetTitle method when it calls the OnLoadPopupContent method from the PopupPM presentation model.

Method Name	Description
SetVisibility	Displays or hides the pop-up object according to state property of the PopupPM presentation model. If the state property is: • **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. The commands are described in the following sections.

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.

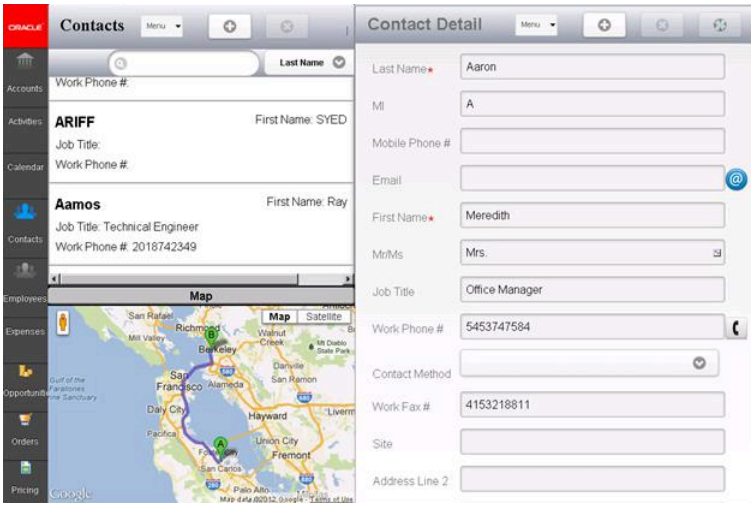
Flow That the GetInlineRoute Method Uses

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.

   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.
8. 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
9. 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.

10. Draws the route from the currentLocation to acctDestination, for example, as shown in the following image.



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.
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 Web template supports list and map rendering. It contains the following code. It uses the jqmMapCtrl div id to render the Google map:

```
<div od-if="Web Engine State Properties, IsMobileApplicationMode">
<div id="SiebelMapContainer" name="SiebelMapContainer"
style="display:none;">
<div id="jqmMapCtrl" name="jqmMapCtrl"></div>
</div>
</div>
```

Methods That Manipulate Property Sets

This topic describes the methods you can use that manipulate property sets. The methods are described in the following subtopics.

Structure of the Property Set

The following table describes the structure of the property set that Siebel Open UI uses in the client.

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.

Property	Description
propArray	Object that contains the values for all properties that the property set contains.
propArrayLen	Length of the propArray property.
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.

11 Reference Information for Siebel Open UI

Reference Information for Siebel Open UI

This chapter describes reference information for Siebel Open UI. It includes the following topics:

- *Life Cycle Flows of User Interface Elements*
- *Notifications That Siebel Open UI Supports*
- *Property Sets That Siebel Open UI Supports*
- *Siebel CRM Events That You Can Use to Customize Siebel Open UI*
- *Languages That Siebel Open UI Supports*
- *Screens and Views That Siebel Mobile Uses*
- *Controls That Siebel Open UI Uses*
- *Browser Script Compatibility*

Life Cycle Flows of User Interface Elements

This topic includes flowcharts that you can use to determine the methods that Siebel Open UI will follow during various steps in the life cycle of a user interface element. The flowcharts are described in the following subtopics.

- *Life Cycle Flows That Save Records*
- *Life Cycle Flows That Handle User Navigation*
- *Life Cycle Flows That Send Notifications*
- *Life Cycle Flows That Create New Records in List Applets*
- *Life Cycle Flows That Handle User Actions in List Applets*

Life Cycle Flows That Save Records

This topic describes the following life cycle flows that Siebel Open UI follows to save records.

- *Flow That Saves Records If the User Uses a Shortcut*
- *Flow That Saves Records If the User Uses the Save Menu*

Flow That Saves Records If the User Uses a_Shortcut

The following figure illustrates the life cycle flow that Siebel Open UI follows 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 steps in the flow, after a user saves a record using shortcut, are as follows:

1. Review A – the A connector connects to the flow described in *Flow That Saves Records If the User Uses the Save Menu*.
2. Process the commands in the Command Manager.

- Run AttachPMBinding("FieldChange") on Presentation Model (PM).
- Run AttachPMBinding("CellChange") on List PM.
- Run the following on the Physical Renderer:

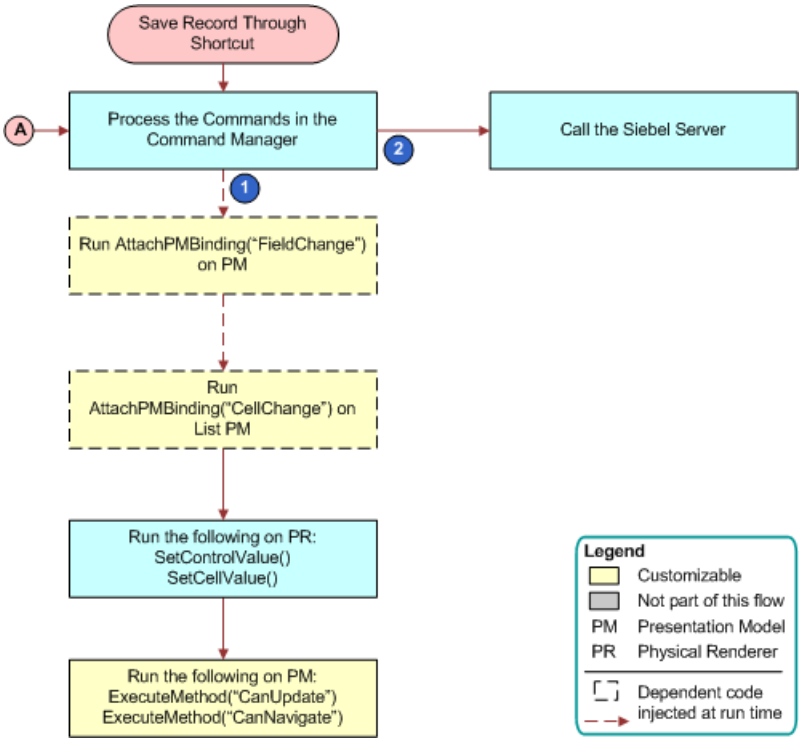
  SetControlValue()

  SetCellValue()
- Run the following on PM:

  ExecuteMethod("CanUpdate")

  ExecuteMethod("CanNavigate")

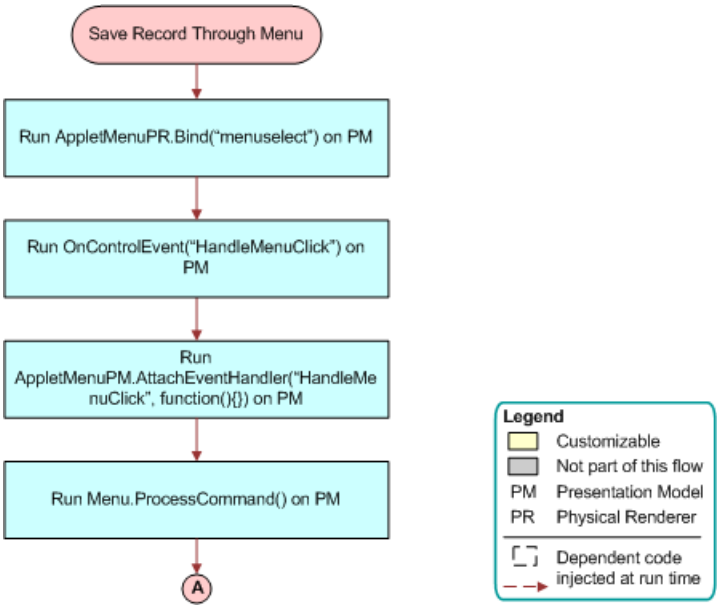
3. Call the Siebel Server.



Flow That Saves Records If the User Uses the Save_Menu

The following figure illustrates the life cycle flow that Siebel Open UI follows to save a record if the user clicks Menu, and then the Save Record menu item. The steps in the flow, after a user saves a record using Menu, are as follows:

1. Run AppletMenuPR.Bind("menuselect") on Presentation Model (PM).
2. Run OnControlEvent("HandleMenuClick") on PM.
3. Run AppletMenuPM.AttachEventHandler("HandleMenuClick", function(){}) on PM.
4. Run Menu.ProcessCommand() on PM.
5. Review A – the A connector connects to the flow described in *Flow That Saves Records If the User Uses a Shortcut*.



Life Cycle Flows That Handle User Navigation

This topic describes the following life cycle flows that Siebel Open UI follows when the user navigates through various items in the client.

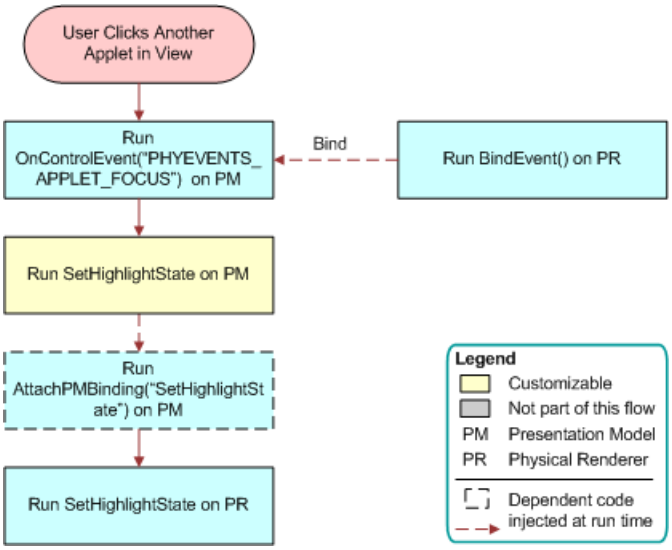
- *Flow That Siebel Open UI Uses If the User Clicks an Applet in a View*
- *Flow That Siebel Open UI Uses If the User Navigates to a View*
- *Flow That Handles Focus Changes in Form Applets*
- *Flow That Handles Focus Changes in List Applets*

Flow That Siebel Open UI Uses If the User Clicks an Applet in a View

The following figure illustrates the life cycle flow that Siebel Open UI follows if the user clicks an applet in a view. The steps in the flow, after a user clicks another applet in a view, are as follows:

1. Run OnControEvent("PHYEVENTS_APPLET_FOCUS") on Presentation Model (PM).

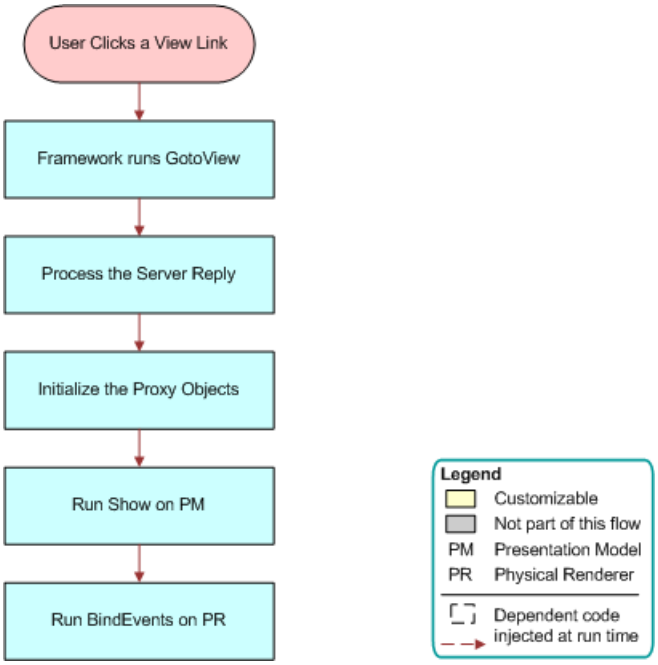
   Run BindEvent() on Physical Renderer (PR).
2. Run SetHighlightState on PM.
3. Run AttachPMBinding("SetHighlightState") on PM.
4. Run SetHighlightState on PR.



Flow That Siebel Open UI Uses If the User Navigates to a View

The following figure illustrates the life cycle flow that Siebel Open UI follows if the user navigates to a view. The steps in the flow, after a user clicks a view link, are as follows:

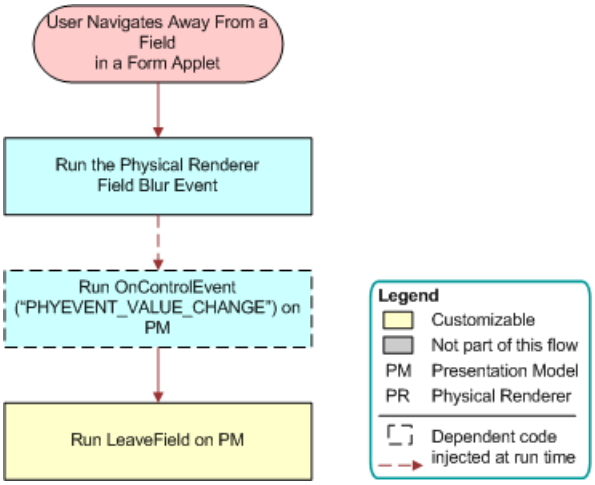
1. Framework runs GoToView.
2. Process the server reply.
3. Initialize the proxy objects.
4. Run Show on Presentation Model.
5. Run BindEvents on Physical Renderer.



Flow That Handles Focus Changes in Form Applets

The following figure illustrates the life cycle flow that Siebel Open UI follows 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. The steps in the flow, after a user navigates away from a field in a form applet, are as follows:

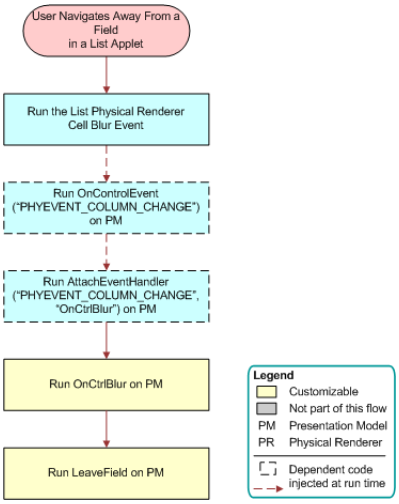
1. Run the Physical Renderer Field Blur event.
2. Run OnControlEvent ("PHYEVENT_COLUMN_CHANGE") on Presentation Model (PM).
3. Run LeaveField on PM.



Flow That Handles Focus Changes in List Applets

The following figure illustrates the life cycle flow that Siebel Open UI follows 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. The steps in the flow, after a user navigates away from a field in a list applet, are as follows:

1. Run the List Physical Renderer Cell Blur event.
2. Run OnControlEvent ("PHYEVENT_COLUMN_CHANGE") on Presentation Model (PM).
3. Run AttachEventHandler ("PHYEVENT_COLUMN_CHANGE", "OnCtrlBlue") on PM.
4. Run OnCtrlBlur on PM.
5. Run LeaveField on PM.



Life Cycle Flows That Send Notifications

This topic describes the following life cycle flows that Siebel Open UI follows to send notifications.

- *Flow That Notifies the Siebel Server*
- *Flow That Sends a Notification State Change*

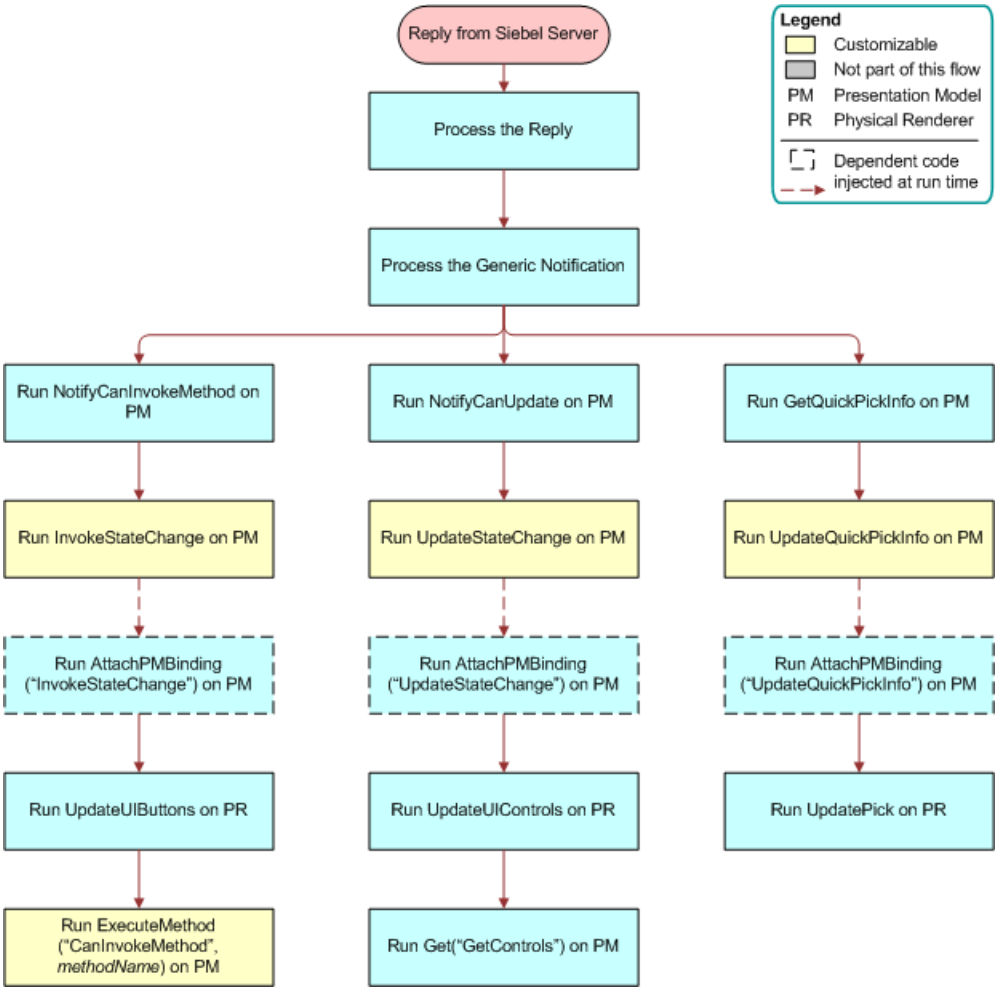
Flow That Notifies the Siebel Server

The following figure illustrates the life cycle flow that Siebel Open UI follows to notify the Siebel Server. The steps in the flow are as follows:

1. Process the reply.
2. Process the generic notification.
    - Run NotifyCanInvokeMethod on Presentation Model (PM) – see step 3.
    - Run NotifyCanUpdateMethod on PM – see step 4.
    - Run GetQuickPickInfo PM – see step 5.
3. Run NotifyCanInvokeMethod on PM.
    - Run InvokeStateChange on PM.
    - Run AttachPMBinding ("InvokeStateChange") on PM.
    - Run UpdateUIButtons on Physical Renderer (PR).
    - Run ExecuteMethod ("CanInvokeMethod", methodName) on PM.
4. Run NotifyCanUpdateMethod on PM.
    - Run UpdateStateChange on PM.
    - Run AttachPMBinding ("UpdateStateChange") on PM.
    - Run UpdateUIControls on PR.

- Run Get("GetControls") on PM.

5. Run GetQuickPickInfo PM.
   - Run UpdateQuickPickInfo on PM.
   - Run AttachPMBinding ("UpdateQuickPickInfo") on PM.
   - Run UpdatePick on PR.



Flow That Sends a Notification State Change

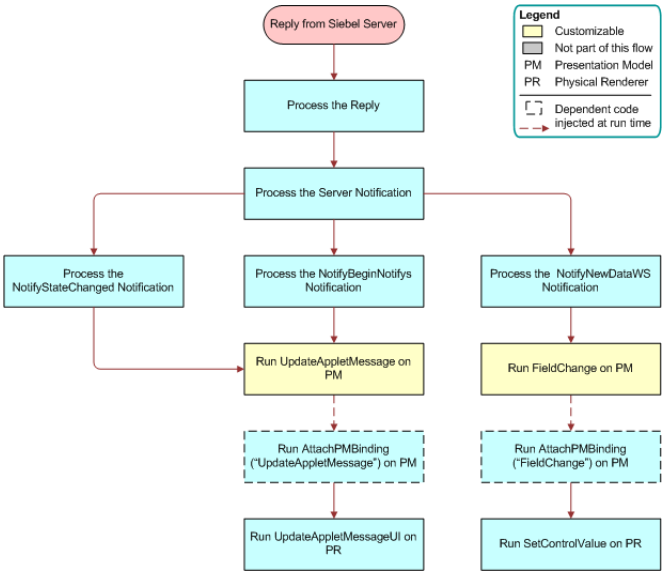
The following figure illustrates the life cycle flow that Siebel Open UI follows to send a notification state change. The steps in the flow, after the user clicks the New button, are as follows:

1. Process the reply.
2. Process the server notification.
   - Process the NotifyStateChanged notification, then go to step 3.
   - Process the NotifyBeginNotifys notification, then go to step 3.
   - Process the NotifyNewDataWS notification, then go to step 4.
3. Run UpdateAppletMessage on Presentation Model (PM).
   - Run AttachPMBinding ("UpdateAppletMessage") on PM.

- Run UpdateAppletMessageUI on PR.

4. Run FieldChange on PM.
   - Run AttachPMBinding ("FieldChange") on PM.
   - Run SetcontrolValue on PR.

For more information about the notifications that this flow describes, see *Notifications That Siebel Open UI Supports*.



Life Cycle Flows That Create New Records in List Applets

This topic describes the following life cycle flows that Siebel Open UI follows to create a new record in a list applet.

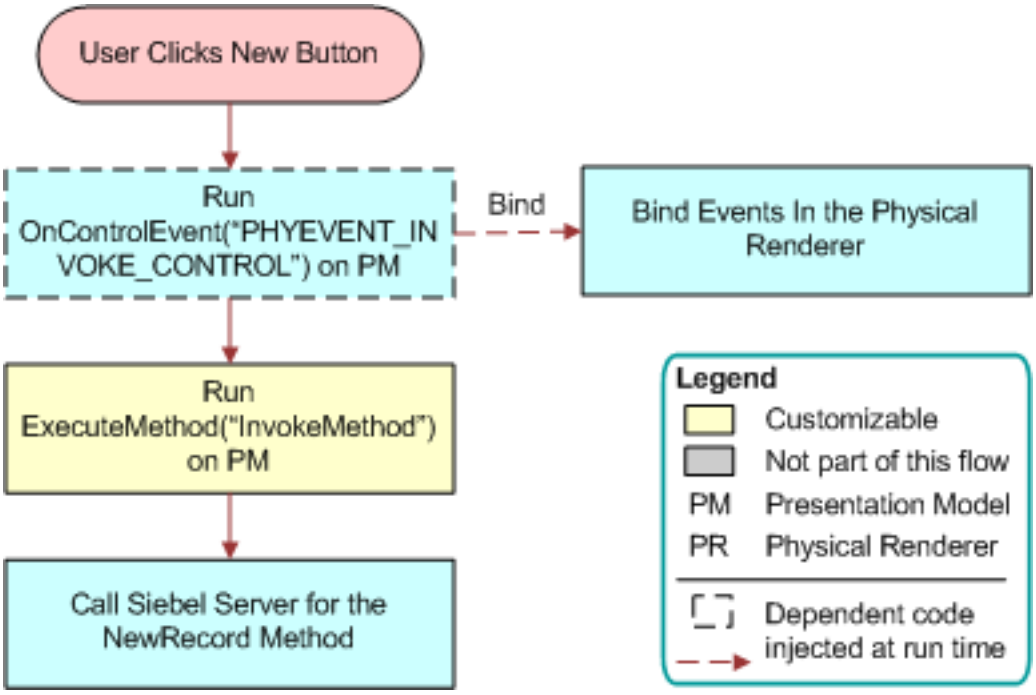
- *Flow That Creates New Records in List Applets, Calling the Siebel Server*
- *Flow That Creates New Records in List Applets, Processing the Server Reply*
- *Flows That Create New Records in List Applets, Updating the User Interface*
- *Flow That Creates New Records in List Applets, Updating the Proxy and Presentation Model*

Flow That Creates New Records in List Applets, Calling the Siebel Server

The following figure illustrates the life cycle flow that Siebel Open UI follows 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. The steps in the flow, after a user clicks the New button, are as follows:

1. Run OnControlEvent("PHYEVENT_INVOKE_CONTROL") on Presentation Model (PM).
   Bind events in the Physical Renderer.
2. Run ExeuteMethod("InvokeMethod") on PM.
3. Call Siebel Server for the NewRecord method.

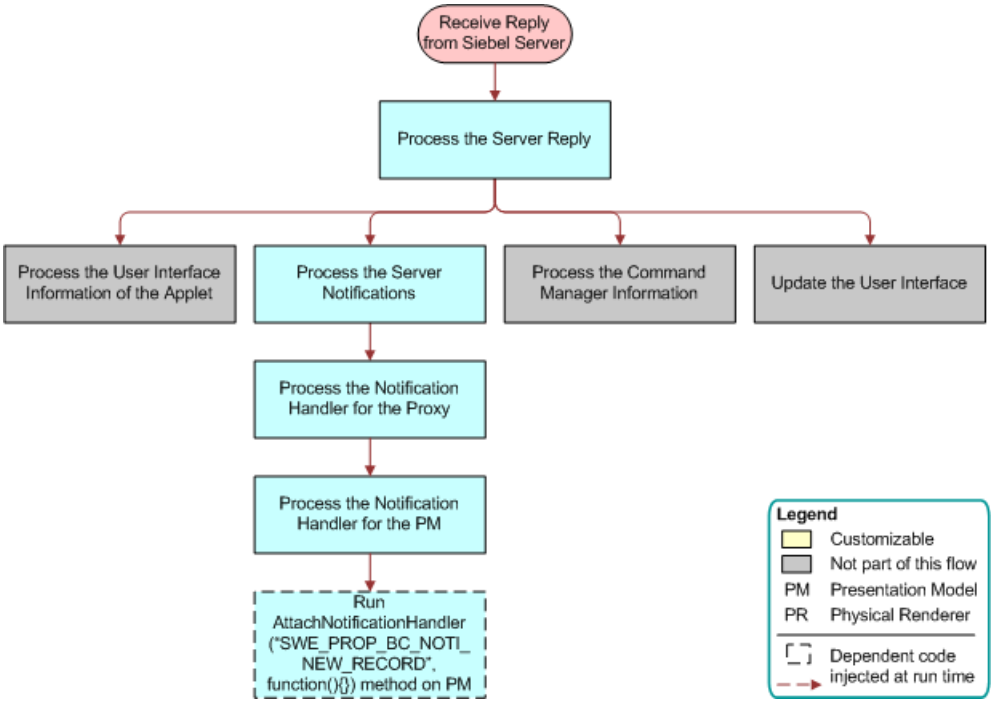
For more information, see *DeleteRecord Method*, *WriteRecord Method*, and *NewRecord Method*.



Flow That Creates New Records in List Applets, Processing the Server Reply

The following figure illustrates the life cycle flow that Siebel Open UI follows 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. The steps in the flow to process the server reply are as follows:

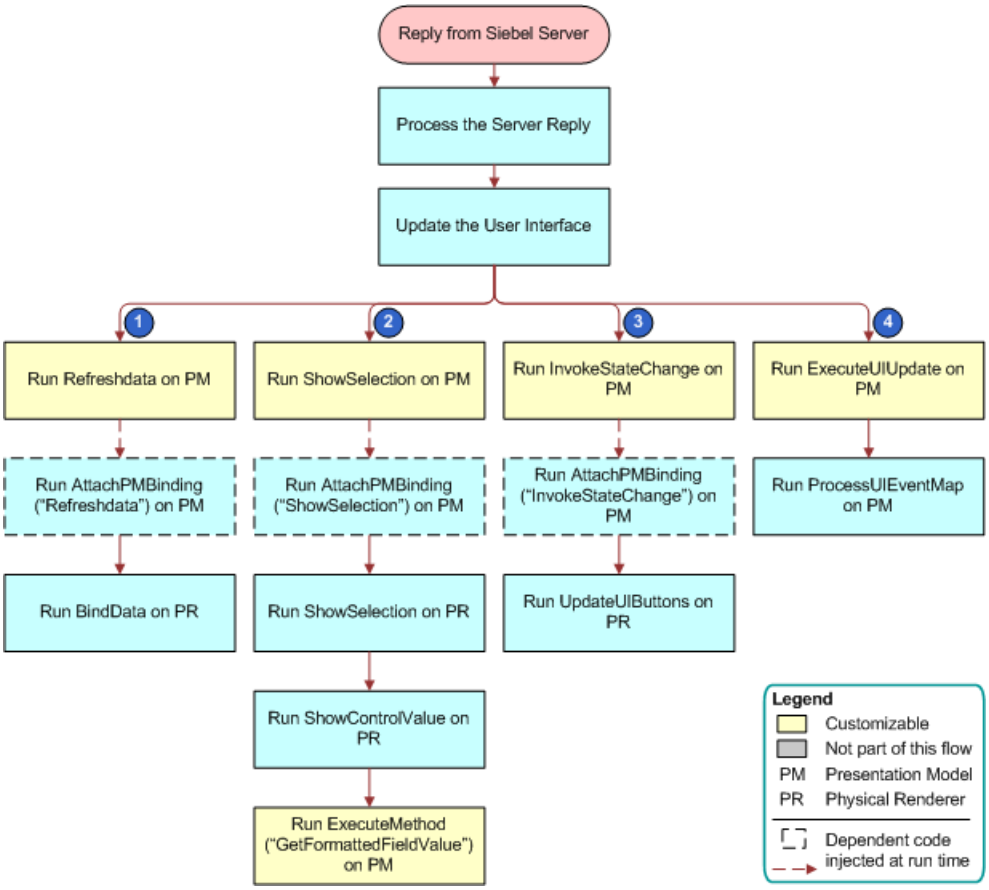
1. Process the User Interface information of the applet.
2. Process the server notifications.
    - Process the notification handler for the proxy.
    - Process the notification handler for the Presentation Model (PM).
    - Run AttachNotificationHandler ("SWE_PROP_BC_NOTI_NEW_RECORD", function(){}) method on PM.
3. Process the command manager information.
4. Update the UI.



Flows That Create New Records in List Applets, Updating the User Interface

The following figure illustrates the life cycle flow that Siebel Open UI follows to update the user interface. The numbers in the figure indicate the sequence that Siebel Open UI uses during this flow. The steps in the flow to update the UI (after a reply from Siebel Server has been processed) are as follows:

1. Run Refreshdata on Presentation Model (PM).
    - Run AttachPMBinding ("Refreshdata") on PM.
    - Run BindData on Physical Renderer (PR).
2. Run Show Selection on PM.
    - Run AttachPMBinding ("ShowSelection") on PM.
    - Run ShowSelection on PR.
    - Run ShowControlValue on PR.
    - Run ExecuteMethod ("GetFormattedFieldValue") on PM.
3. Run InvokeStateChange on PM.
    - Run AttachPMBinding ("InvokeStateChange") on PM.
    - Run UpdateUIButtons on PR.
4. Run ExecuteUIUpdate on PM.
    - Run ProcessUIEventMap on PM.



Flow That Creates New Records in List Applets, Updating the Proxy and Presentation Model

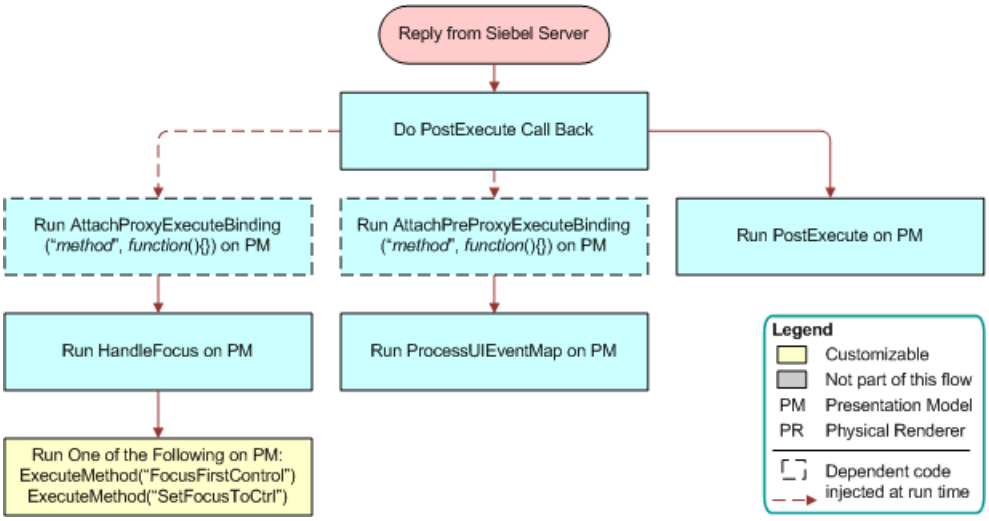
The following figure illustrates the life cycle flow that Siebel Open UI follows to update the proxy and presentation model. The steps in the flow are as follows:

1. Reply from Siebel Server.
2. Do PostExecute Call Back.
    - Run AttachProxyExecuteBinding ("method", function(){}) on Presentation Model (PM).

      Run Handle Focus on PM

      Run one of the following on PM: ExecuteMethod("FocusFirstControl") or ExecuteMethod("SetFocusToCtrl")
    - Run AttachPreProxyExecuteBinding ("method", function(){}) on Presentation Model.

      Run ProccessUIEventMap on PM.
    - Run PostExecute on PM.



Life Cycle Flows That Handle User Actions in List Applets

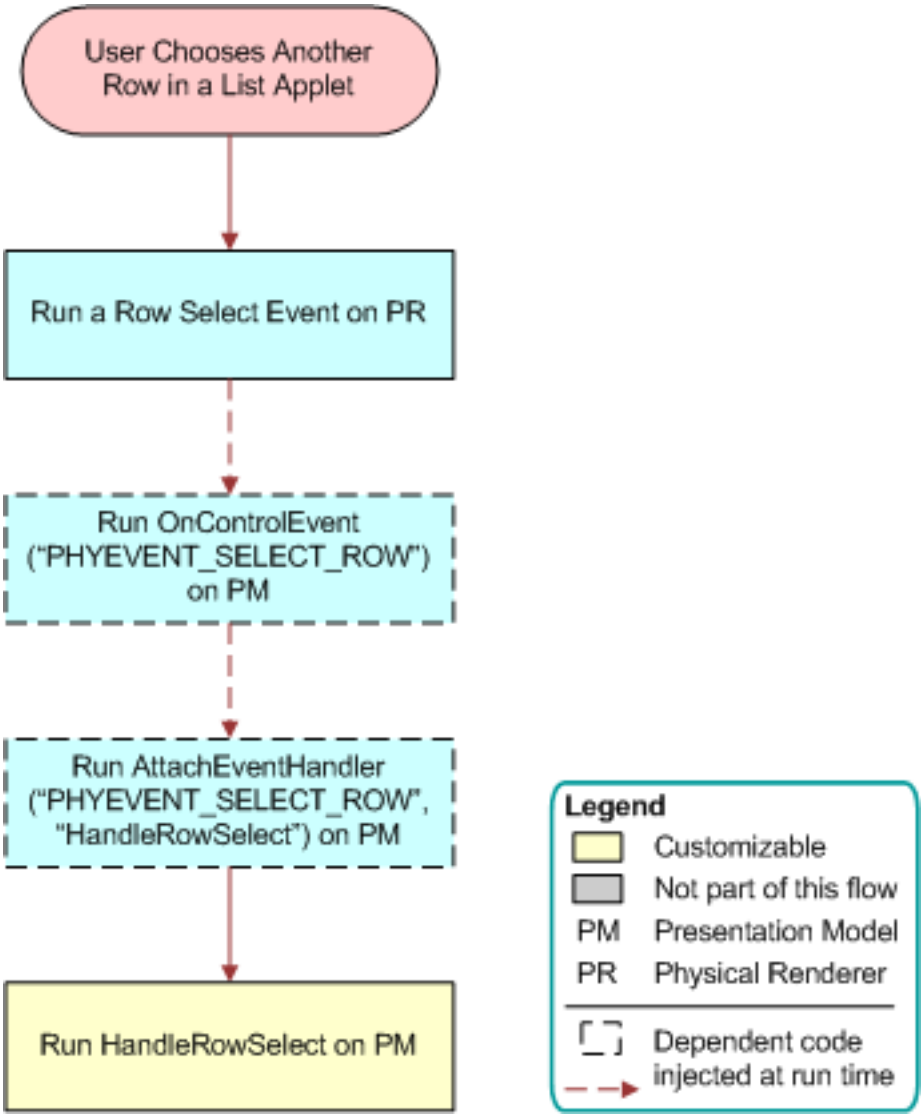
This topic describes the following life cycle flows that Siebel Open UI follows depending on the chosen user action in a list applet.

- *Flow That Handles Navigation to Another Row in List Applets*
- *Flow That Handles the Pagination Button in List Applets*
- *Flow That Handles a Column Sort in List Applets*
- *Flow That Handles a Cell Click in List Applets*
- *Flow That Handles a Cell Edit and Blur in List Applets*
- *Flow That Handles a Drilldown in List Applets*

Flow That Handles Navigation to Another Row in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user navigates to another row in a list applet. The steps in the flow are as follows:

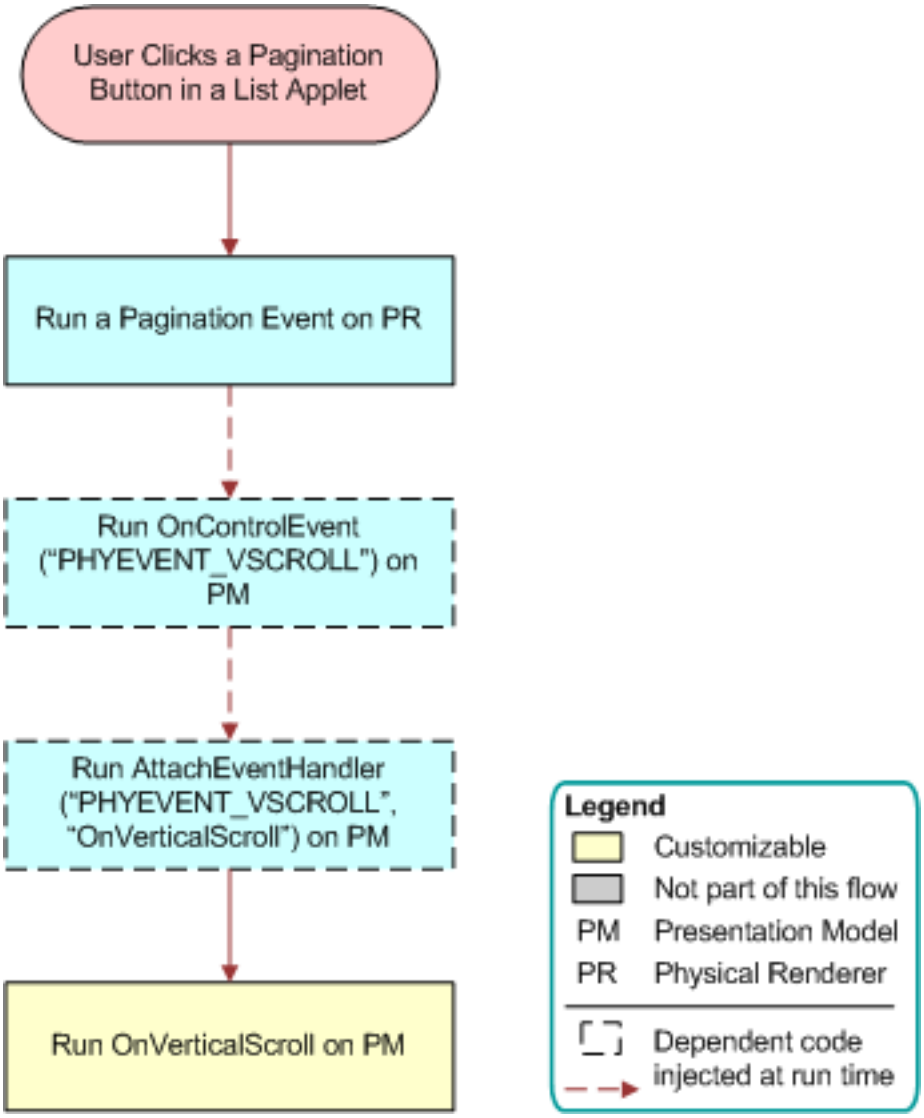
1. User selects another row in a list applet.
2. Run a Row Select Event on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_SELECT_ROW") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_SELECT_ROW", "HandleRowSelect") on Presentation Model.
5. Run HandleRowSelect on PM.



Flow That Handles the Pagination Button in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user clicks the pagination button in a list applet. The steps in the flow are as follows:

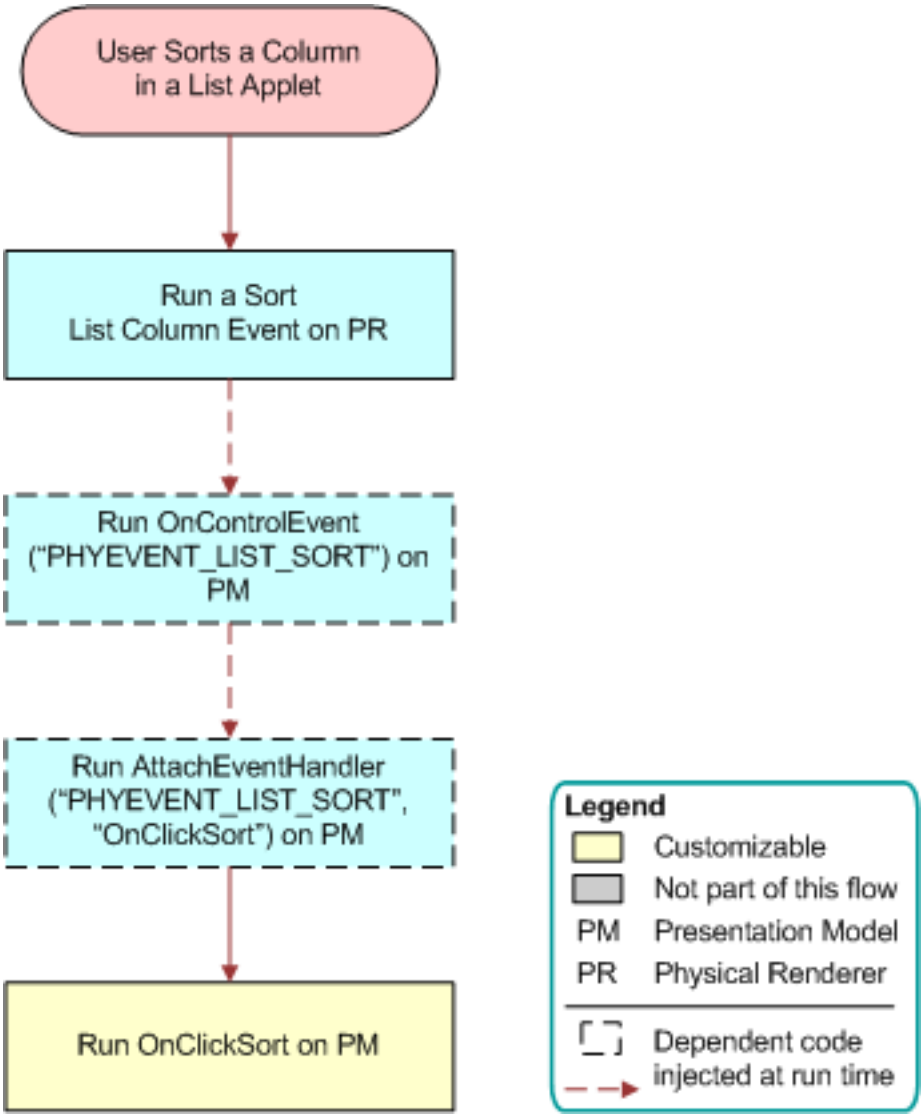
1. User clicks a Pagination button in a list applet.
2. Run a Pagination Event on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_VSROLL") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_VSCROLL", "OnVerticalScroll") on Presentation Model.
5. Run OnVerticalScroll on PM.



Flow That Handles a Column Sort in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user sorts a column in a list applet. The steps in the flow are as follows:

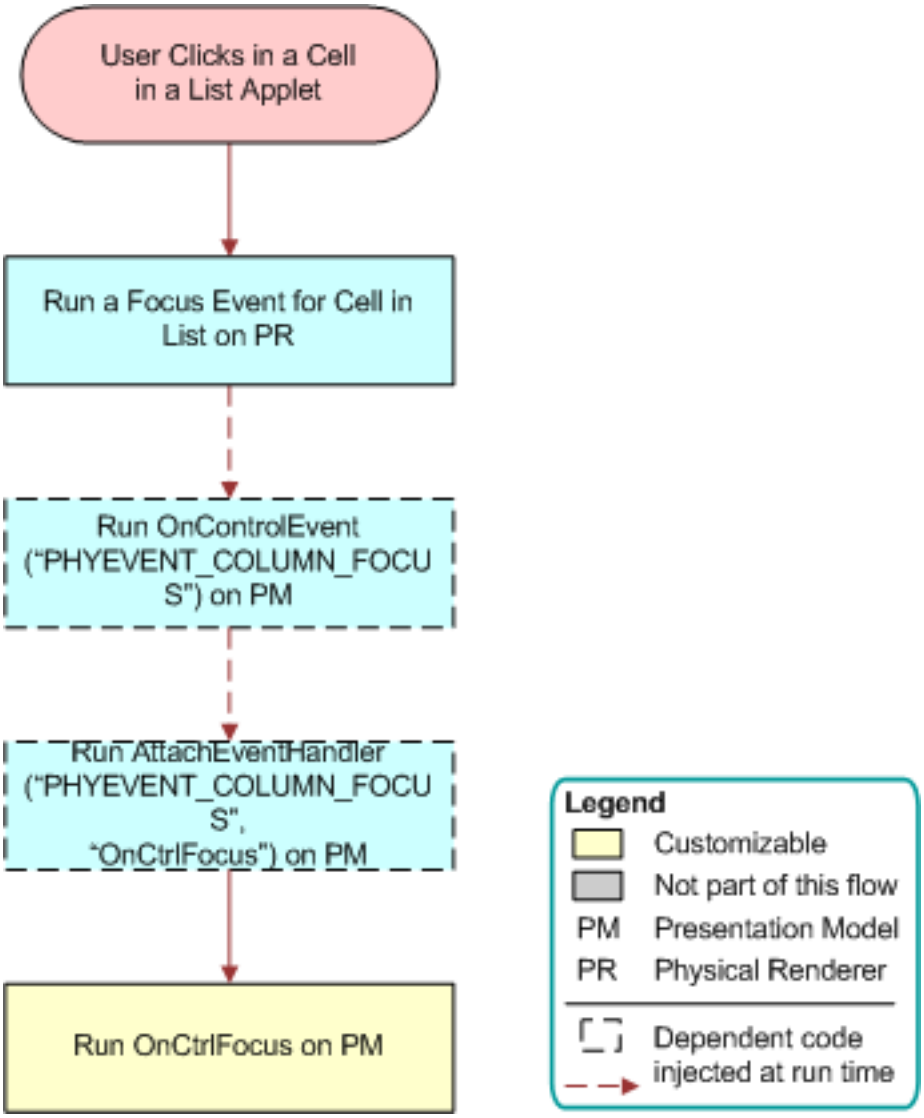
1. User sorts a column in a list applet.
2. Run a Sort List Column Event on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_LIST_SORT") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_LIST_SORT", "OnClickSort") on Presentation Model.
5. Run OnClickSort on PM.



Flow That Handles a Cell Click in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user clicks a cell in a list applet. The steps in the flow are as follows:

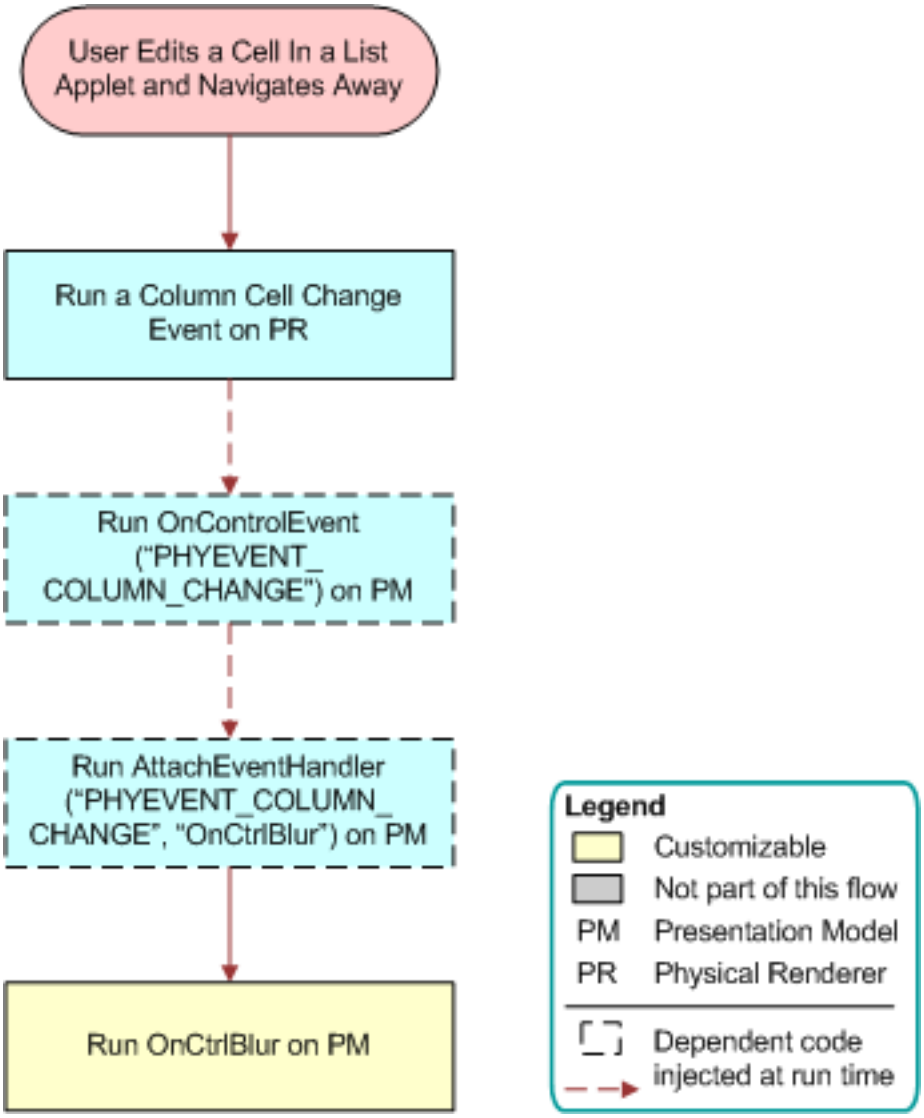
1. User clicks in a cell in a list applet.
2. Run a Focus Event for Cell in List on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_COLUMN_FOCUS") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_COLUMN_FOCUS", "OnCtrlFocus") on Presentation Model
5. Run OnCtrlFocus on PM.



Flow That Handles a Cell Edit and Blur in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user edits a cell in a list applet, and then navigates away from this cell. The steps in the flow are as follows:

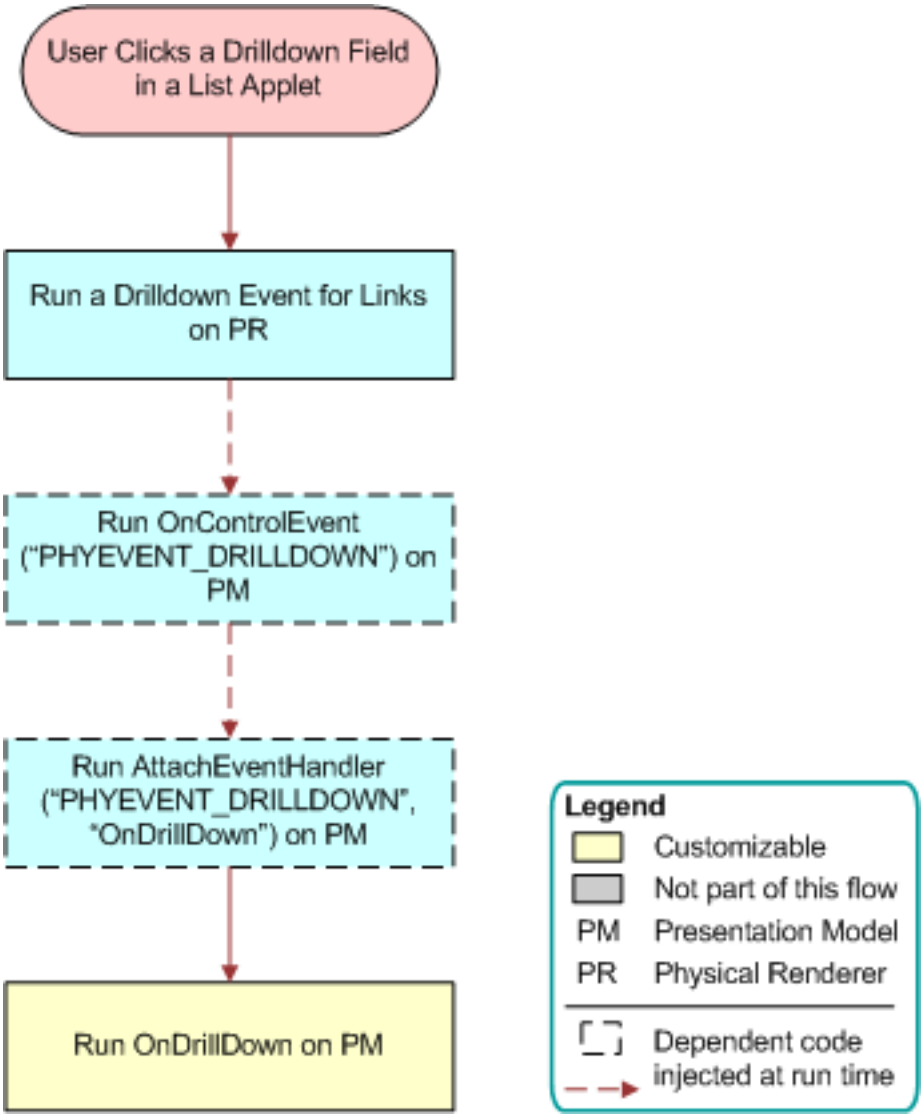
1. User edits a cell in a list applet and navigates away.
2. Run a Column Cell Change Event on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_COLUMN_CHANGE") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_COLUMN_CHANGE", "OnCtrlBlur") on Presentation Model.
5. Run OnCtrlBlur on PM.



Flow That Handles a Drilldown in List Applets

The following figure illustrates the flow that Siebel Open UI follows if the user clicks a drilldown field in a list applet. The steps in the flow are as follows:

1. User clicks a drilldown field in a list applet.
2. Run a drilldown event for links on Physical Renderer.
3. Run OnControlEvent ("PHYEVENT_DRILLDOWN") on Presentation Model.
4. Run AttachEventHandler ("PHYEVENT_DRILLDOWN", "OnDrillDown") on Presentation Model.
5. Run OnDrillDown on PM.



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*
- *Using Notifications with Operations That Call Methods*
- *Notify Generic Notification Type*
- *NotifyStateChanged Notification Type*
- *Example Usages of Notifications*

For more information about configuring Siebel Open UI to use notifications, see *AttachNotificationHandler Method*.

Summary of Notifications That Siebel Open UI Supports

The following table describes the notification types that Siebel Open UI supports. For more information, see *New Notification User Interfaces*.

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	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: • state • value Siebel Open UI can provide summary or detailed state information. For more information, see *NotifyStateChanged Notification Type* .
NotifyGeneric	SWE_PROP_BC_NOTI_GENERIC	Identifies the predefined and custom notifications that the Siebel application must send. Siebel Open UI addresses most predefined generic notifications to a particular applet. 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.
NotifyNewSelection	SWE_PROP_NOTI_SELECTED	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: • 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 You cannot use NotifyNewSelection with a multi-select. You can use the following syntax in the property set that Siebel Open UI sends: **`SWE_PROP_BC_NOTI_ACTIVE_ROW = index`** **`SWE_PROP_NOTI_SELECTED = Boolean`** where:

Notification	Type	Description
		• index identifies the index of the row that Siebel Open UI is activating or deactivating. • Boolean is true or false.
NotifyNewActiveRow	SWE_PROP_BC_NOTI_ NEW_ACTIVE_ROW	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. You can use the following syntax: `SWE_PROP_BC_NOTI_ACTIVE_ROW = row` where: • row identifies the row that Siebel Open UI is activating or deactivating.

NotifyDeleteRecord	SWE_PROP_BC_NOTI
	
	_DELETE_RECORD
		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.
		
		You can use the following syntax in the property set that Siebel Open UI sends:
		
		`SWE_PROP_BC_NOTI_ACTIVE_ROW = index`
		`bUp = Boolean`
		
		where:
		
		• index identifies the index of a record that resides in the current set of records that Siebel Open UI is deleting.
		• Boolean is one of the following values:
		• **true.** Shift records up after the delete.
		• **false.** Shift records down after the delete.
		For an example usage of this notification, see *Customizing the Presentation Model to Handle Notifications*.

NotifyDeleteRecordSet	SWE_PROP_BC_NOTI
	
	_DELETE_WORKSET
		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.
		
		You can use the following syntax in the property set that Siebel Open UI sends:
		
		`index:index`
		`NumRows/nr:number`
		
		where:
		
		• index identifies the start index of the record that Siebel Open UI is deleting.

Notification	Type	Description
		• number identifies the number of rows that Siebel Open UI must delete. For more information, see *Using Notifications with Operations That Call Methods*.

NotifyInsertWorkSet	SWE_PROP_BC_NOT
	I_INSERT_WORKSET
		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.
		You can use the following syntax in the property set that Siebel Open UI sends:
		`index:index_value`
		`SWE_FIELD_VALUE_STR:child`
		`SWE_PROP_VALUE_ARRAY:array`
		where:
		• index identifies the index of the record that Siebel Open UI is inserting.
		• child identifies the child property set that contains the record data.
		• array 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.
		For more information, see *Using Notifications with Operations That Call Methods*.

NotifyNewData	SWE_PROP_BC _NOTI_NEW_DATA	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.
NotifyNewPrimary	SWE_PROP_BC_ NOTI_NEW_PRIMARY	Sets the primary record in a multi-value group. The RepopulateField notification calls NotifyNewPrimary.

NotifyNewRecord	SWE_PROP_BC_
	NOTI_NEW_RECORD
		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:
		`SWE_PROP_BC_NOTI_ACTIVE_ROW = index`
		`bInsertBefore = Boolean`
		where:
		• row identifies the index of the record that Siebel Open UI is creating.
		• Boolean is one of the following values:
		• **true.** Place the new record before the previous active row.

Notification	Type	Description
		• **false.** Place the new record after the previous active row. For a similar usage of this notification, see *Customizing the Presentation Model to Handle Notifications*.
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	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. You can use the following syntax in the property set that Siebel Open UI sends: `SWE_PROP_NOTI_FIELD = field` where: • field identifies the name of the field that Siebel Open UI is modifying.

NotifyNewDataWorkset	SWE_PROP_BC_NOTI_
	NEW_DATA_WS
		Notifies the client business component of a field value that Siebel Open UI modified for a field that resides on the Siebel Server.
		You can use the following syntax in the property set that Siebel Open UI sends:
		`SWE_PROP_NOTI_FIELD = field`
		`SWE_PROP_FIELD_VALUES = child`
		where:
		• field identifies the name of the field that Siebel Open UI is modifying.
		• child identifies the name of the child property set that contains the modification details.
		You can use the following syntax in the child property set:
		`SWE_PROP_FIELD_ARRAY:string1`
		`SWE_PROP_VALUE_ARRAY:string2`
		where:
		• string1 is an encoded string that identifies the field index.

Notification	Type	Description
		• string2 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.
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	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: • Using a default query specification • Starting or running a query This situation can occur through a predefined or custom configuration, or in reply to a query that the user performs. You can use the following syntax in the property set that Siebel Open UI sends: `SWE_PROP_NOTI_FIELD = fieldSWE_PROP_VALUE = search specification"` where: • field identifies the name of the field that Siebel Open UI is querying. • search specification identifies a query specification that is defined on this field.

Notification	Type	Description
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.
NotifyExecute	SWE_PROP_BC _NOTI_EXECUTE	Notifies the client business component that Siebel Open UI is running a business component on the Siebel Server. You can use the following syntax in the property set that Siebel Open UI sends: `srt = sort specifications` where: • sort specification identifies the sort specification that Siebel Open UI runs. • search specification identifies the search specification that Siebel Open UI runs.
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

The following table 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.

Sub Type	Description
SWEICanInvokeMethod	Enables the refresh button.

Sub Type	Description
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 *private field* is a type of field that only allows the record owner to view the record. For more information, see *Siebel Object Types Reference* .
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	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:
	`argsArray[0] = applet name`
	`argsArray[1-x] = value`
	where:
	• *applet* name identifies the name of the applet where Siebel Open UI sends the notification.
	• *value* is one of the following:
	○ **1.** Indicates selected.
	○ **0.** Indicates not selected.

SWEAInvokeMethod	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.
	You can use the following syntax in the decoded array:
	`argsArray[0] = applet`
	`argsArray[1] = method`
	where:
	• *applet* identifies the name of the applet that Siebel Open UI calls during the first operation.
	• *method* identifies the name of the subsequent method that Siebel Open UI calls.
	For more information, see *InvokeMethod Method for Presentation Models*.

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.
GetQuickPickInfo	Sends the values of a picklist to an applet. You can use the following syntax in the decoded array:

Sub Type	Description
	argsArray[0] = placeholder **argsArray[1] = view** **argsArray[2] = applet** **argsArray[3] = identifier** **argsArray[4] = control** **argsArray[5-x] = string** where: • placeholder is a placeholder array that you can you use. • view identifies the name of the view where Siebel Open UI displays the picklist. • applet identifies the name of the applet where Siebel Open UI displays the picklist. • identifier identifies the HTML identifier of the control that requested the picklist values. • control contains one of the following values: o **true.** Picklist is associated with the control. o **false.** Picklist is not associated with the control. • string contains the values of the picklist.
BegRow	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: **argsArray[0] = applet name** **argsArray[1] = value** where: • applet name identifies the name of the applet where Siebel Open UI sends the notification. • value 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.
CloseCurrencyPickApplet	Sends a notification to close a currency applet.
ClosePopup	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: **argsArray[0] = applet** where: • applet identifies the name of the applet.

NotifyStateChanged Notification Type

The following table describes the subtypes of the NotifyStateChanged type.

Sub Type	Description
activeRow	Identifies the active row of the business component. You can use ar (active row) to abbreviate activeRow.
bCanDelete	Returns a Boolean value that includes one of the following values: • **0.** The business component can delete a field. • **1.** The business component cannot delete a field. You can use cd (can delete) as an abbreviation for bCanDelete.
bCanInsert	Returns a Boolean value that includes one of the following values: • **0.** The business component can insert a field. • **1.** The business component cannot insert a field. You can use ci (can insert) as an abbreviation for bCanInsert.
bCanInsertDynamic	Returns a Boolean value that includes one of the following values: • **0.** The business component can insert a dynamic field. • **1.** The business component cannot insert a dynamic field. You can use cud (can insert dynamic) as an abbreviation for bCanInsertDynamic.
bCanMergeRecords	Returns a Boolean value that includes one of the following values: • **0.** Merge is available in multi select mode. • **1.** Merge is not available in multi select mode. You can use cm (can merge) as an abbreviation for bCanMergeRecords.
bCanQuery	Returns a Boolean value that includes one of the following values: • **0.** The business component can query a field. • **1.** The business component cannot query a field. You can use cq (can query) as an abbreviation for bCanQuery.
bCanUpdate	Returns a Boolean value that includes one of the following values: • **0.** The business component can update a field. • **1.** The business component cannot update a field. You can use cu (can update) as an abbreviation for bCanUpdate.
bCanUpdateDynamic	Returns a Boolean value that includes one of the following values: • **0.** The business component can update a dynamic field. • **1.** The business component cannot update a dynamic field. You can use cud (can update dynamic) as an abbreviation for bCanUpdateDynamic.
bCommitPending	Returns a Boolean value that includes one of the following values:

Sub Type	Description
	• **0.** A commit is pending on the business component. • **1.** A commit is not pending on the business component. You can use cp (commit pending) as an abbreviation for bCommitPending.
bDelRecPending	Returns a Boolean value that includes one of the following values: • **0.** A delete is pending on the business component. • **1.** A delete is not pending on the business component. You can use dp (delete pending) as an abbreviation for bDelRecPending.
bExecuted	Returns a Boolean value that includes one of the following values: • **0.** Siebel Open UI finished processing the business component records. • **1.** Siebel Open UI did not finish processing the business component records. You can use **ex** (executed) as an abbreviation for bExecuted.
bHasAssocList	Determines whether or not the business component is an association business component. An *association business component* is a type of business component that includes an intertable.
bInMultiSelMode	Returns a Boolean value that includes one of the following values: • **0.** The business component is in multiselect mode. • **1.** The business component is not in multiselect mode. You can use **ms** (multiselect) as an abbreviation for bInMultiSelMode.
bInQueryState	Returns a Boolean value that includes one of the following values: • **0.** The business component is in a query state. • **1.** The business component is not in a query state. You can use **qs** (query state) as an abbreviation for bInQueryState.
bInverseSelection	Returns a Boolean value that includes one of the following values: • **0.** An inverse of selection is occurring on the business component. • **1.** An inverse of selection is not occurring on the business component. You can use**is** (inverse selection) as an abbreviation for bInverseSelection.
bNewRecPending	Returns a Boolean value that includes one of the following values: • **0.** A new record is pending on the business component. • **1.** A new record is not pending on the business component. You can use **np** (new record pending) as an abbreviation for bNewRecPending.
bNotifyEnabled	Returns a Boolean value that includes one of the following values:

Sub Type	Description
	• **0.** The business component is not enabled for notifications. • **1.** The business component is enabled for notifications. You can use **n** (enabled) as an abbreviation for bNotifyEnabled.
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 Usages of Notifications

This topic describes example usages of notifications.

Example of the NotifyBeginNotifys Notification

The following code is an example usage of the NotifyBeginNotifys notification:

```
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_DELETE_RECORD"),
function (propSet){
// Change has occurred at server BC. Do something here:
this.SetProperty ("Refresh_Renderer", true);
});
```

Example of the NotifyNewSelection Notification

The following code is an example usage of the NotifyNewSelection notification:

```
this.AttachNotificationHandler(consts.get("SWE_PROP_NOTI_SELECTED"), function
(propSet){
if (propSet.GetProperty(consts.get("SWE_PROP_NOTI_SELECTED")) === "false")
this.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.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 Notifications

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.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.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.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"){
 // The dropdown contains a value that we don't like..
 // Let us disable it.
 this.SetProperty ("disablePick", true);
 }
 }
});
```

Example of the ClosePopup Notification

The following code is an example usage of the ClosePopup notification:

```
this.AttachNotificationHandler(consts.get("SWE_PROP_BC_NOTI_GENERIC"), function
(propSet){
var type = propSet.GetProperty(consts.get("SWE_PROP_NOTI_TYPE"));
if (type === "ClosePopup"){
// The following is an example. PM's should not alert anything. Leave that to the PRs.
alert ("Make sure you have collected all details. You next operation will save the
record!");
}
});
```

Example of the SWEAInvokeMethod Notification

The following code is an example usage of the SWEAInvokeMethod notification:

```
this.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.SetProperty ("isChained", true);
}
}
});
```

Example of the NotifyStateChanged Notification

The following code is an example usage of the NotifyStateChanged notification:

```
this.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.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.SetProperty ("AlertUser", true);
}
}
...
...
});
```

Property Sets That Siebel Open UI Supports

The following table 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.

Property Set	Description
SWE_PROP_NC_PDQ_INFO	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: • **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	Defines the beginning of the visibility property set. It uses the following properties: • **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. You can use the VisDropDownItem property as the predefined bubbled handler to do user interface binding.
SWE_PROP_NC_SCREENCTRL _INFO	Defines information for the first level in a screen.

Property Set	Description
	It uses the 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. You can use the GetTabInfo property as the predefined bubbled handler.
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: • **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: • **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 bubble handlers: • **GetDataReloadDirty.** Specifies the flag property. • **GetSelectedTabKey.** Selection of tabs. • **GetSelectedTabLinkKey.** Selection of links underneath tabs. • **GetTabInfo.** All tabs. • **GetTabLinkInfo.** All links.

Siebel CRM Events That 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 mouse down, mouse over, 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*.

- You can use JavaScript methods to manage Siebel CRM events, such as BindEvent, OnControlEvent, and so on. For more information, see *OnControlEvent Method*, *BindEvents Method* and *AttachEventHandler Method*.

Events That You Can Use to Customize Form Applets

The following table describes the events that you can use to customize a form applet.

Event	Description

PHYEVENT_APPLET_FOCUS	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:
	
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_APP LET_FOCUS""),`
	`function () {`
	`// My applet recived focus.`
	`this.SetProperty (""AppletInFocus"", true);`
	`return (true);`
	`});"`

PHYEVENT_CONTROL_FOCUS	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:
	
	• **control.** The control object that receives focus.
	• **value.** The value of the control object that receives focus.
	The following code is an example usage of this event:
	
	`this.AttachEventHandler(siebConsts.get("PHYEVENT_CONTR OL_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 ("FlaggedControlsValue");`
	`if (value > maxValue){`
	`// Value higher than allowed when receiving focus. Let's flag this.`
	`this.SetProperty ("FlagHigher", true);`
	`}`
	`}`
	`}`
	`return (true);`
	`});`

PHYEVENT_CONTROL_BLUR	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: • **control.** The control object that lost focus. • **value.** The value of the control object that lost focus. The following code is an example usage of this event:

Event	Description

	"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
	(""FlaggedControlsValue"");
	if (valueObject[control.GetName()] !== value){
	// Value change. Need to update internal
	storage, and fire any potential extensions attached to
	the property.
	valueOjbect[control.GetName()] = value;
	this.SetProperty (""FlaggedControlsValue"",
	valueObject);
	}
	}
	}
	return (true);
	});"

PHYEVENT_INVOKE_CONTROL	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:
	• **method.** The method that Siebel Open UI calls.
	• **inputPS.** The input property set that Siebel Open UI sends to the method that it calls.
	The following code is an example usage of this event:
	"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_CONTROL""),
	function (method, inputPS, ai) {
	if (method === ""WriteRecord""){
	var valueObject = this.GetProperty
	(""FlaggedControlsValue"");
	var min = this.Get(""MinRangeForFlagged"");
	for (var value in valueObject){
	if (value < min){
	alert (""Invalid Values. Think again!"");
	return (false);
	}
	}
	}
	return (true);
	});"

PHYEVENT_INVOKE_PICK	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: • **control.** The control object of the picklist that Siebel Open UI called. The following code is an example usage of this event: "this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_MVG""),

Event	Description

	function (control) {
	if (control === this.GetProperty(""AddressMVG"")){
	varhighValue = this.Get(""HighVal"");
	if (highValue < this.Get (""MinRangeForFlagged""){
	alert (""Sorry, can't popup this MVG"");
	return (false);
	}
	}
	return (true);
	});"

PHYEVENT_INVOKE_MVG	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:
	• **control.** The control object of the multivalue group that Siebel Open UI called.
	The following code is an example usage of this event:
	"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INV
	OKE_MVG""),
	function (control) {
	if (control === this.GetProperty(""AddressMVG"")){
	varhighValue = this.Get(""HighVal"");
	if (highValue < this.Get (""MinRangeForFlagged""){
	alert (""Sorry, can't popup this MVG"");
	return (false);
	}
	}
	return (true);
	});"

PHYEVENT_INVOKE_DETAIL	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:
	• **control.** The control object of the pop-up details control that Siebel Open UI called.
	The following code is an example usage of this event:
	"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INV
	OKE_DETAIL""),
	function (control) {
	if (control === this.GetProperty(""City"")){
	varhighValue = this.Get(""HighVal"");
	if (highValue < this.Get (""MinRangeForFlagged""){
	alert (""Sorry, can't use this Details"");
	return (false);
	}
	}
	return (true);
	});"

PHYEVENT_INVOKE_EFFDAT	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: • **control.** The control object of the effective dating pop-up control that Siebel Open UI called.

Event	Description

	The following code is an example usage of this event:
	
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_EFFDAT""),`
	`function (control) {`
	`if (control === this.GetProperty(""AccountTrail"")){`
	`varhighValue = this.Get(""HighVal"");`
	`if (highValue < this.Get (""MinRangeForFlagged""){`
	`alert (""Sorry, can't open this Dating Popup"");`
	`return (false);`
	`}`
	`}`
	`return (true);`
	`});"`

PHYEVENT_INVOKE_COMBO	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:
	
	• **control.** The control object of the combo box or dropdown list that Siebel Open UI called.
	
	The following code is an example usage of this event:
	
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_COMBO""),`
	`function (control) {`
	`if (control === this.GetProperty(""Designation"")){`
	`varhighValue = this.Get(""HighVal"");`
	`if (highValue < this.Get (""MinRangeForFlagged""){`
	`alert (""Sorry, can't open this Dropdown"");`
	`return (false);`
	`}`
	`}`
	`return (true);`
	`});"`

PHYEVENT_INVOKE_CURRENCY	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:
	
	• **control.** The control object of the currency calculator that Siebel Open UI called.
	
	The following code is an example usage of this event:
	
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_CURRENCY""),`
	`function (control) {`
	`if (control === this.GetProperty(""RevenueControl"")){`
	`varhighValue = this.Get(""HighVal"");`
	`if (highValue < this.Get (""MinRangeForFlagged""){`
	`alert (""Sorry, can't open this currency field"");`
	`return (false);`
	`}`
	`}`
	`return (true);`
	`});"`

Event	Description
PHYEVENT_INVOKE_TOGGLE	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: • **value.** Contains the value of the toggle control that exists after the user action. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_INVOKE_TOGGLE""), function (value) { if (value === this.GetProperty(""FlaggedControl"")){ varhighValue = this.Get(""HighVal""); if (highValue < this.Get (""MinRangeForFlagged""){ alert (""Sorry, change the Range value to toggle""); return (false); } } return (true); });"`
PHYEVENT_DRILLDOWN_FORM	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: • **control.** Identifies the control object that contains the destination of the drilldown control. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_DRILLDOWN_FORM""), function (control) { if (control === this.GetProperty(""AccountDrill"")){ varhighValue = this.Get(""HighVal""); if (highValue < this.Get (""MinRangeForFlagged""){ alert (""Sorry, change the Range value to drilldown""); return (false); } } return (true); });"`
PHYEVENT_ENTER_KEY_PRESS	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: • **control.** Identifies the control object where the user used the Enter key. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_ENTER_KEY_PRESS""), function (control) { if (control === this.GetProperty(""Salary"")){ varhighValue = this.Get(""HighVal""); if (highValue < this.Get (""MinRangeForFlagged""){ alert (""Sorry, change the Range value to`

Event	Description
	commit""); return (false); } } return (true); });"

PHYEVENT_ESC_KEY_PRESS	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:
	• **control.** Identifies the control object where the user used the Esc key.
	The following code is an example usage of this event:
	"this.AttachEventHandler(siebConsts.get(""PHYEVENT_ESC_KEY_PRESS""),
	function (control) {
	if (control === this.GetProperty(""Salary"")){
	varhighValue = this.Get(""HighVal"");
	if (highValue < this.Get (""MinRangeForFlagged""){
	alert (""Sorry, change the Range value to undo"");
	return (false);
	}
	}
	return (true);
	});"

PHYEVENT_TAB_KEY_PRESS	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:
	• **control.** Identifies the control object where the user used the Tab key.
	The following code is an example usage of this event:
	"this.AttachEventHandler(siebConsts.get(""PHYEVENT_TAB_KEY_PRESS""),
	function (control) {
	if (control === this.GetProperty(""Salary"")){
	varhighValue = this.Get(""HighVal"");
	if (highValue < this.Get (""MinRangeForFlagged""){
	alert (""Sorry, change the Range value to undo"");
	return (false);
	}
	}
	return (true);
	});"

Events That You Can Use to Customize List Applets

The following table describes the events that you can use to customize a list applet.

Event	Description

PHYEVENT_SELECT_ROW	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.
	
	You can use this event with the following objects:
	
	• **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.
	The following code is an example usage of this event:
	
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_SELECT_ROW""),`
	`function (rowIndex, shiftKey, controlKey) {`
	`if (rowIndex === 0 && shiftKey){`
	`alert (""Do not multiselect all rows."");`
	`return (false);`
	`}`
	`});"`

PHYEVENT_COLUMN_FOCUS	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:
	
	• **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.
	The following code is an example usage of this event:
	
	`"this.AttachEventHandler(siebConsts.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 (""FlaggedControlsValue"");`
	`if (value > maxValue){`
	`// Flag value that is higher than allowed when receiving focus.`
	`this.SetProperty (""FlagHigher"", true);`
	`}`
	`}`
	`}`
	`return (true);`
	`});"`

PHYEVENT_COLUMN_BLUR	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:

Event	Description
	• **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. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.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 (""FlaggedControlsValue""); if (value > maxValue){ // Value higher than allowed when receiving focus. Let's flag this. this.SetProperty (""FlagHigher"", true); } } } return (true); });"`
PHYEVENT_DRILLDOWN_LIST	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: • **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. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_DRILLDOWN_LIST""), function (colName, rowIndex) { if (name === ""Type"")){ varmaxOptyArray = this.Get(""mO""); if (maxOptyArray[rowIndex] > this.Get(""HigVal"")){ alert (""Fix opportunity value before drilldown.""); return (false); } } return (true); });"`
PHYEVENT_VSCROLL_LIST	Siebel Open UI triggers the PHYEVENT_VSCROLL_LIST event when the user clicks the next page or previous 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:

Event	Description
	• **direction.** Includes one the following values that describes the type of pagination that the user attempted: o **PAG_NEXT_RECORD.** (Prior to Siebel CRM 17.0 only) User paginated to the next record. o **PAG_PREV_RECORD.** (Prior to Siebel CRM 17.0 only) User paginated to the previous record. o **PAG_NEXT_SET.** Displays the next record in the set. o **PAG_PREV_SET.** Displays the previous record in the set. o **PAG_FIRST_SET.** Displays the first record in the set. o **PAG_LAST_SET.** Displays the last record in the set. o **PAG_SCROLL_UP.** User scrolled up. o **PAG_SCROLL_DN.** User scrolled down. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_VSCROL L_LIST""), function (direction) { if (direction ===SiebelApp.Contants.Get(""PAG_NEXT_SET"")){ alert (""Jump record by record. Not sets.""); return (false); } return (true); });"`
PHYEVENT_SORT_LIST	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: • **colName.** Identifies the name of the column that the user is sorting. • **direction.** Identifies one of the following directions: o **SORT_ASCENDING.** The user is sorting the column in ascending order. o **SORT_DESCENDING.** The user is sorting the column in descending order. The following code is an example usage of this event: `"this.AttachEventHandler(siebConsts.get(""PHYEVENT_SORT_L IST""), function (colName, direction) { if (colName === ""Type""){ if (direction === SiebelApp.Constants.Get(""SORT_ASCENDING"")){ alert (""You cannot sort this column.""); return (false); } } return (true); });"`
PHYEVENT_HIER_EXPAND	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:

Event	Description

	• **rowNum.** Contains the row number that the user is expanding. It uses 0 as the number for the first row.
	The following code is an example usage of this event:
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_HIER_EXPAND""),`
	`function (rowNum) {`
	`if (rowNum === 0)){`
	`varhighValue = this.Get(""HighVal"");`
	`if (highValue < this.Get (""MinRangeForFlagged""){`
	`alert (""Sorry. Change the Range value to expand this row."");`
	`return (false);`
	`}`
	`}`
	`return (true);`
	`});"`

PHYEVENT_HIER_COLLAPSE	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:
	• **rowNum.** Contains the row number that the user is collapsing. It uses 0 as the number for the first row.
	The following code is an example usage of this event:
	`"this.AttachEventHandler(siebConsts.get(""PHYEVENT_HIER_COLLAPSE""),`
	`function (rowNum) {`
	`if (rowNum === 2)){`
	`varhighValue = this.Get(""HighVal"");`
	`if (highValue < this.Get (""MinRangeForFlagged""){`
	`alert (""Sorry, change the Range to collapse this row."");`
	`return (false);`
	`}`
	`}`
	`return (true);`
	`});"`

Languages That Siebel Open UI Supports

This topic lists the languages that Siebel Open UI supports. It supports the i18N internationalization standard. In most situations, it does not hard code strings, and it uses language independent values for each LOV (list of values) instead of translated values. For more information about how to customize Siebel Open UI to support multiple languages, see *Displaying Control Labels in Different Languages*.

Languages That Siebel Open UI Supports for Windows, AIX, Oracle Solaris, and HP-UX

The following table lists the languages That Siebel Open UI supports for Windows, AIX, Oracle Solaris, and HP-UX. The Lang column lists the *language_code*.

Lang	Codepage	Windows	AIX	Oracle Solaris	HP-UX
ARA	1256	Arabic (Saudi Arabia)	AR_SA.UTF-8	ar_SA.UTF-8	ar_SA. utf8
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_UR.utf8
HEB	1255	Hebrew	HE_IL.UTF-8	he_IL.UTF-8	univ.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

Lang	Codepage	Windows	AIX	Oracle Solaris	HP-UX
TRK	1254	Turkish	TR_TR.UTF-8	tr_TR.UTF-8	tr_TR.utf8
PLK	1250	Polish	PL_PL.UTF-8	pl_PL.UTF-8	pl_PL.utf8

Languages That Siebel Open UI Supports for Linux RH, Linux SuSe, Enterprise Linux, and Java Locale Code

The following table lists the languages that Siebel Open UI supports for Linux RH, Linux SuSe, Enterprise Linux, and Java Locale Code. The Lang column lists the *language_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

Lang	Codepage	Linux RH	Linux SuSe	Enterprise Linux	Java Locale Code
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*
- *Screens and Views That Siebel Sales Uses*
- *Screens and Views That Siebel Service Uses*
- *Screens and Views That Siebel Pharma Uses*

Screens and Views That Siebel Consumer Goods Uses

The following table lists the predefined screens and views that Siebel Mobile uses for Siebel Consumer Goods.

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

Screen	View Name in Client	View Name in Siebel Tools
	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
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

The following table lists the predefined screens and views that Siebel Mobile uses for Siebel Sales.

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
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

Screen	View Name in Client	View Name in Siebel Tools
	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

The following table lists the predefined screens and views that Siebel Mobile uses for Siebel Service.

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

Screen	View Name in Client	View Name in Siebel Tools
	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
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

The following table lists the predefined screens and views that Siebel Mobile uses for Siebel Pharma.

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
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

Screen	View Name in Client	View Name in Siebel Tools
	(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*
- *Other Controls That Siebel Open UI Uses*

Predefined Controls That Siebel Open UI Uses

The following table describes the 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
btnControl	Displays a button.	phyrenderer.js
chartControl	Displays a chart.	
checkBoxCtrl	Displays a check box.	
comboControl	Displays a combobox.	

Control Name	Description	Where Defined
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.	siebelmaprenderer.js
Tiles UI	Displays records in a tile.	Tilescrollcontainer.js

Other Controls That Siebel Open UI Uses

The following table describes other controls that Siebel Open UI uses. It uses all these controls in the controlbuilder.js file in addition to the files that the Where Defined column lists.

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

Control Name	Description	Where Defined
RTCeditor	Enters rich text.	ckeditor_3.6.3.zip ckeditor-custom.zip
FileUploader	Uploads files.	jquery.fileupload.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 *Application Programming Interface*.

Sequence That Siebel Open UI Uses 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 calss 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 *Deriving Presentation Models, Physical Renderers, and Plug-in Wrappers*.

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 Siebel Runtime Repository. 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 Siebel Runtime Repository. 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

The following table describes the event handlers that you can use in browser script to handle a predefined event for a Siebel object type.

Object Type	Event Handler
Application	You can use the following event handlers: • Application_InvokeMethod • Application_PreInvokeMethod
Applet	You can use the following event handlers: • Applet_ChangeFieldValue • Applet_ChangeRecord • Applet_InvokeMetohd • Applet_PreInvokeMethod • Applet_Load
Business component	You can use the following event handler: • BusComp_PreSetFieldValue
Business service	You can use the following event handlers:

Object Type	Event Handler
	• Service_InvokeMethod • Service_PreCanInvokeMethod • Service_PreInvokeMethod

Event Handlers You Can Use to Handle Predefined DOM Events

The following table describes the event handlers that you can use in browser script to handle a predefined DOM event for a Siebel control object type.

Object Type	Event Handler
Control in the Siebel Open UI client.	You can use the following event handlers: • OnBlur • OnFocus

Methods You Can Use in Browser Script

The following table describes the methods that you can use in a browser script for each Siebel object type that Oracle's Siebel Open UI can use.

Object Type	Method
Applet	You can use the following methods: • ActiveMode • BusComp • BusObject • FindControl • InvokeMethod • Name • ReInit
Application	You can use the following methods: • FindApplet • ActiveApplet • ActiveViewName • ActiveBusObject

Object Type	Method
	• ActiveBusComp • FindBusObject • GetProfileAttr • GetService • InvokeMethod • IsReady • Name • NewPropertySet • SWEAlert • ShowModalDialog • SeblTrace
Business Component	You can use the following methods: • BusObject • GetFieldValue • GetFormattedFieldValue • GetSearchExpr • GetSearchSpec • InvokeMethod • Name • SetFieldValue • SetFormattedFieldValue • WriteRecord
Business Object	You can use the following methods: • FirstBusComp • GetBusComp • Name • NextBusComp
Business Service	You can use the following methods: • InvokeMethod • Name • SetProperty • PropertyExists • RemoveProperty • GetProperty • GetFirstProperty • GetNextProperty

Object Type	Method
Control	You can use the following methods: • Applet • BusComp • GetValue • Name • SetValue • SetReadOnly • SetEnabled • SetVisible • SetProperty • GetLabelProperty • GetProperty • SetLabelProperty
Property Set	You can use the following methods: • AddChild • Copy • GetChild • GetChildCount • GetFirstProperty • GetNextProperty • GetProperty • GetPropertyCount • GetType • GetValue • InsertChildAt • PropertyExists • RemoveChild • RemoveProperty • Reset • SetProperty • SetType • SetValue

12 Post-Upgrade Configuration Tasks

Post-Upgrade Configuration Tasks

This chapter describes post-upgrade configuration tasks. Depending on the customization of your deployment, these tasks might be required after upgrading to the latest Siebel CRM release from releases prior to Siebel Innovation Pack 2014. This appendix includes the following topics:

- *Updating Physical Renderer Customizations for Controls*
- *Modifying Physical Renderer Code for Event Helper*
- *Overriding Plug-In Wrappers*

Updating Physical Renderer Customizations for Controls

This topic describes how to work with existing customizations of physical renderers. It contains the following information:

- *Control DOM Access and Changes*
- *Control Value Access and Changes*
- *Control State Manipulation*

Control DOM Access and Changes

Siebel Open UI uses plug-in wrappers to oversee controls and their Document Object Model (DOM) manipulations. Any renderer code required to work with control level DOM elements defers to its respective control plug-in wrapper interface to get the DOM representation.

Any changes required to the DOM will need to be completed by way of the control plug-in wrapper interface. The renderer must not make the changes in itself. The plug-in wrapper should be decorated with the ability to do what is required on the physical UI based on the logical control that it is representing.

To adhere to conventions in current releases, you might need to determine if you have code that needs to be modified. To do this, you must find the control DOM access with specific types in your custom renderer code, and replace that code with new code.

To find and modify the control DOM access types in your custom renderer code

1. Determine if you have code that needs to be modified by searching for calls similar to either of the following code samples:
   - `var controElement = $('[name="' + control.GetInputName() + '"]');`
   - `var controElement = $('#' + control.GetInputName());`

> **Note:** Access is not limited to these calls. Similar types of calls that attempt to find the DOM element using the control object should also be replaced.

2. Replace all instances of calls similar to code discovered in Step 1 using the following convention:

```
var controElement = this.GetUIWrapper(control).GetEl();
```

   These modifications help ensure that the correct jQuery element representing the control on the UI is retrieved, irrespective of the type of renderer from which the call is being made.

`GetUIWrapper` is a plug-in builder API that is injected into all physical renderers. It returns the plug-in wrapper of the control object that is passed to it. There are various APIs, such as GetEl(), that are executable in the wrapper. For more information about these plug-in wrappers, see *Plug-in Wrapper Class*.

Control Value Access and Changes

Siebel Open UI requires that renderer code that retrieves and sets control values from the DOM consults with the plug-in wrapper interface of the control. Any changes required to the DOM will need to be completed by way of the control plug-in wrapper interface.

Any changes required to the value of the controls will need to be completed by way of the control plug-in wrapper interface. The renderer must not make the changes in itself. The plug-in wrapper must be decorated with the ability to do what is required on the physical UI based on the logical control that it is representing. Wrapper methods can further be customized to decorate on top of these values if required.

To adhere to the conventions of current releases, you might need to determine if you have code that needs to be modified. To do this, you must find the control value access with specific types in your custom renderer code, and replace that code with new code.

To find and modify the control value access types in your custom renderer code

1. Determine if you have code that needs to be modified by searching for calls similar to either of the following code samples:

   - `var value = $('[name="' + control.GetInputName() + '"]').val();`
   - `var value = $('#' + control.GetInputName()).attr('val');`

   > **Note:** Access is not limited to these calls. Similar types of calls that attempt to find the DOM element using the control object should also be replaced.

2. Replace all instances of calls similar to code discovered in Step 1 using following convention:

```
var value = this.GetUIWrapper(control).GetValue();
```

   These modifications help ensure that the correct jQuery element representing the control on the UI is retrieved, irrespective of the type of renderer from which the call is being made.

   `GetValue()` is the plug-in wrapper API that is responsible for getting the DOM value of the control. Similar to this explanation, this change will first fetch the correct wrapper for the control in question and then executes the GetValue API

3. Determine if you have code that needs to be modified by searching for calls similar to either of the following code samples:
   - `$('[name="' + control.GetInputName() + '"]').val(newValue);`
   - `$('#' + control.GetInputName()).attr('val', newValue);`

   Note: Access is not limited to these calls. Similar types of calls that attempt to find the DOM element using the control object should also be replaced.
4. Replace all instances of calls similar to code discovered in Step 3 using following convention:

   ```
   this.GetUIWrapper(control).SetValue(value, index);
   ```

   Note: The value set affects neither the client record set nor the server data, unless explicitly committed.
5. (Optional) Other customizations might be necessary if you are using a custom plug-in wrapper that overrides the base wrapper's API which may affect the value before setting it on the DOM. The following is an example of such a customization:

   ```
   CustomPW.prototype.SetValue = function (value, index) {
   value = value + "_suffix";
   SiebelAppFacade.CustomPW.superclass.SetValue.call(this, value, index);
   }
   ```

Control State Manipulation

The manipulation of the DOM state of a control occurs in a single call in the control's wrapper element using the SetState API. In some earlier releases, this type of manipulation could have been done in many different ways. Therefore, you might need to locate and modify any outdated custom renderer code.

To find and modify the control state manipulations in your custom renderer code

1. Determine if you have code similar to the following:

   ```
   $('[name="' + control.GetInputName() + '"]').hide();
   ```
2. Replace all instances of calls similar to code discovered in Step 1 by a call to the control's plug-in wrapper that internally affects the state of the element and hides it. Use the following code as guidance:

   ```
   this.GetUIWrapper(control).SetState(consts.get("SHOW"), false);
   ```
3. Determine if you have code similar to the following:

   ```
   $('[name="' + control.GetInputName() + '"]').attr("readOnly", "readOnly");
   ```

   The code here is a case where a particular control is being made non-editable on the DOM.
4. Replace all instances of calls similar to code discovered in Step 3 using following convention:

   ```
   this.GetUIWrapper(control).SetState(consts.get("EDITABLE"), false);
   ```
5. Determine if you have code similar to the following:

   ```
   $('[name="' + control.GetInputName() + '"]').focus();
   ```

The code here is a case where there is an attempt to set focus on a particular control.

6. Replace all instances of calls similar to code discovered in Step 5 using following convention:

```
this.GetUIWrapper(control).SetState(consts.get("FOCUS"), true);
```

The SetState API exists in the prototype space of the plug-in wrapper and can also be overridden in a custom plug-in wrapper to be used to affect the functionality of setting states on the control.

For more information about state modification, including parameters accepted by SetState, and the modifications made to the control element, see *Architecture of Siebel Open UI* and *Application Programming Interface*.

Modifying Physical Renderer Code for Event Helper

This topic describes how to work with previously customized physical renderer code that deal specifically with event binding. It describes the changes that might be required to allow this code to work in current releases. It contains the following information:

- *Binding Stray DOM Events*
- *Binding Events for Controls*

Binding Stray DOM Events

The Event Helper object manages events and their handlers. This object is available as the custom physical renderer for event handler management. You can use the helper object to attach custom event handlers to stray DOM objects. The objects fall into one of the following categories:

- **DOM elements configured in SWE OUI templates.** These are DOM elements configured in the SWE OUI Templates but not in the repository, and are directly addressed and manipulated at the client-level using JavaScript code.
- **DOM elements with no representation.** These are DOM elements that have no representation on the server, and are completely constructed and manipulated at the client-level using JavaScript code.

The event helper object can attach and manage event handlers to both types of DOM elements previously listed. However, it is essential that any stray binding occurring in the custom renderer code is modified to work with the Event Helper object. The Event Helper object homogenizes events between different platforms and devices, and consequently, bound handlers are consistently run across devices.

Modifications Required for DOM Elements Configured in SWE OUI Templates

You must find and modify the instances of DOM elements configured in SWE OUI templates in your custom physical renderer code. The ID or the name used to find the DOM will have been configured as a placeholder using a SWE OUI Template file.

To bind DOM elements configured in SWE OUI templates

1. Determine if you have custom physical renderer code similar to the following:

```
CustomPhysicalRenderer.prototype.BindEvents = function(){
$("[id=" + "customdiv" + "]").bind("click", { ctx : this }, function(event){
event.data.ctx.GetPM().OnControlEvent(("DIV_CLOSE"));
});
```

```
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
};
```

This event will be attached to a handler in a custom presentation model using an `AttachEventHandler` call. The call will then trigger custom functionality when the handler is run. A possible outcome of this handler is to affect a model property, which would subsequently be latched on to by a PM binding in the renderer that would hide or remove the div element.

2. Modify the code located in Step 1 to resemble the following code:

```
CustomPhysicalRenderer.prototype.BindEvents = function () {
var closeElement = $("[id=" + "customdiv " + "]"),
eventHelper = SiebelApp.S_App.PluginBuilder.GetHoByName("EventHelper");
if (eventHelper && closeElement.length) {
eventHelper.Manage(closeElement, "click", { ctx: this }, OnClickDiv);
}
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
};
function OnClickDiv(event) {
event.data.ctx.GetPM().OnControlEvent(consts.get("DIV_CLOSE"));
}
```

> **Note:** The code in this example is only meant as a guide.

This click event in this code is attached to the stray DOM element using the Manage API of the Event Helper object. The click is homogenized to work with touch based events on touch based devices. OnClickDiv is the handler that is passed.

For the full list of parameters that the Manage API uses, see *Application Programming Interface*.

Modifications Required for DOM Elements with No Representation

For DOM elements with no representation, you must find and modify these instances your custom physical renderer for code. These are typically found where the DOM is being created and objects are being attached. This will usually have no representation anywhere in the SWE server.

To bind DOM elements with no representation

1. Determine if you have custom physical renderer code similar to the following:

```
CustomPhysicalRenderer.prototype.ShowUI = function () {
var clientHTML = "<div id='moreinfo'>Click here for more information about Customer
Types</div>",
appletContainer = this.GetPM().Get("GetFullId");
$("#" + appletContainer).append(clientHTML);
SiebelAppFacade.CustomPhysicalRenderer.superclass.ShowUI.call(this);
};
CustomPhysicalRenderer.prototype.BindEvents = function(){
$("[id=" + "moreinfo" + "]").bind("mouseover", { ctx : this }, function(event){
vent.data.ctx.GetPM().OnControlEvent("MORE_INFO");
});
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
```

```
};
```

The first section of the code creates a client side piece of the DOM which is appended to the end of the applet container in the ShowUI section of the custom renderer. The BindEvents section is then overridden to attach a custom event handler to the DOM element.

The second section of the code is an event that will be attached to a custom presentation model using an `AttachEventHandler` call. The call will then trigger custom functionality when the handler is run. This will display a dialog containing additional information about the contextual record.

2. Modify the code located in Step 1 to resemble the following code:

```
CustomPhysicalRenderer.prototype.BindEvents = function () {
var moreInfoElement = $("[id=" + "moreinfo" + "]"),
eventHelper = SiebelApp.S_App.PluginBuilder.GetHoByName("EventHelper");
if (eventHelper && moreInfoElement.length) {
eventHelper.Manage(moreInfoElement, "mouseover", { ctx: this }, OnClickMoreInfo);
}
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
};
function OnClickMoreInfo(event) {
event.data.ctx.GetPM().OnControlEvent("MORE_INFO");
}
```

Note: The code in this example is only meant as a guide.

The mouseover event is attached to the stray DOM element that has been created in ShowUI using the Manage API of the Event Helper object. Mouseover is homogenized to work with touch based events on touch based devices, if available.

Binding Events for Controls

This topic describes how to bind events for controls. Similarly to stray DOM event binding, any renderer code that bound events on to existing repository based controls will now have to work with the Event Helper object to bind handlers to controls instead of direct jQuery calls, which is necessary to homogenize events bound to controls across different platforms.

To bind control events

1. Determine if you have custom renderer code that contains control event binding of the following types:

```
CustomPhysicalRenderer.prototype.BindEvents = function () {
var controlSet = this.GetPM().Get("GetControls");
for (var controlName in controlSet) {
if (controlSet.hasOwnProperty(controlName)) {
var control = controlSet[controlName];
if (control.GetName() === "Probability") {
$('[name="' + control.GetInputName() + '"]').on("click", { ctx: this }, function () {
event.data.ctx.GetPM().OnControlEvent(("PROBABLITY_CLICK"));
});
}
}
}
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
```

```
};
```

In this code, the complete set of controls is obtained from the framework PM property and the render is looping through set. Upon encountering a particular control by the repository name `Probability`, a click event handler is being bound to the DOM element representing that control which then triggers the event `PROBABILITY_CLICK` on to the PM. Eventually, a custom handler is attached as a customized presentation model using the `AttachEventHandler` API.

2. Modify the code located in Step 1 to resemble the following code:

```
CustomPhysicalRenderer.prototype.BindEvents = function () {
var eventHelper = SiebelApp.S_App.PluginBuilder.GetHoByName("EventHelper"),
controlSet = this.GetPM().Get("GetControls"),
controlEl = null;
for (var controlName in controlSet) {
if (controlSet.hasOwnProperty(controlName)) {
var control = controlSet[controlName];
if (control.GetName === "Probability") {
controlEl = this.GetUIWrapper(control).GetEl();
if (controlEl.length && eventHelper) {
eventHelper.Manage(controlEl, "click", { ctx: this }, OnClickProbability);
}
}
}
}
SiebelAppFacade.CustomPhysicalRenderer.superclass.BindEvents.call(this);
};
function OnClickProbablility(){
event.data.ctx.GetPM().OnControlEvent(("PROBABLITY_CLICK"));
}
```

In this new code, if the correct control is found using the matching condition, first, the control's element is obtained using the GetEl() API of the control's wrapper. This is subsequently used in the Manage() API of the Event Helper to attach a homogenized click handler on to the DOM element of the control. The named method OnClickProbability is triggered when the event occurs on the control.

Overriding Plug-In Wrappers

This topic describes how to create and apply plug-in wrappers for customization of control appearance and behavior. It contains the following information:

- *About Overriding Plug-In Wrappers*
- *Overview of the Skeleton Structure of a Plug-in Wrapper*
- *About Presentation Model-Injected APIs in Plug-in Wrappers*

About Overriding Plug-In Wrappers

Plug-in wrappers provide an effective manner in which control objects can be customized. Plug-in wrappers effectively manage the entire life cycle of an individual control, including but not limited to its DOM creation, appearance, behaviors and states. For more information about plug-in wrappers, see *About Plug-in Wrappers*.

In some older releases, physical renderers were responsible for control behavior. This meant that control representation and its lifecycle were tightly coupled with the physical renderer in which it was hosted. The introduction of plug-in wrappers decouples the control representation and its lifecycle.

Consequently, the physical renderer is no longer aware of the type of controls it needs to deal with, but rather talks to the plug-in wrapper of the control(s) that it hosts. The actual action will take place inside the plug-in wrapper. The result is that the plug-in wrapper is not bound to any particular applet or its physical renderer.

The topics that follow describe how to override plug-in wrappers. For more information about the API specification and the inheritance hierarchy of the plug-in wrappers, see *Architecture of Siebel Open UI*.

Overview of the Skeleton Structure of a Plug-in Wrapper

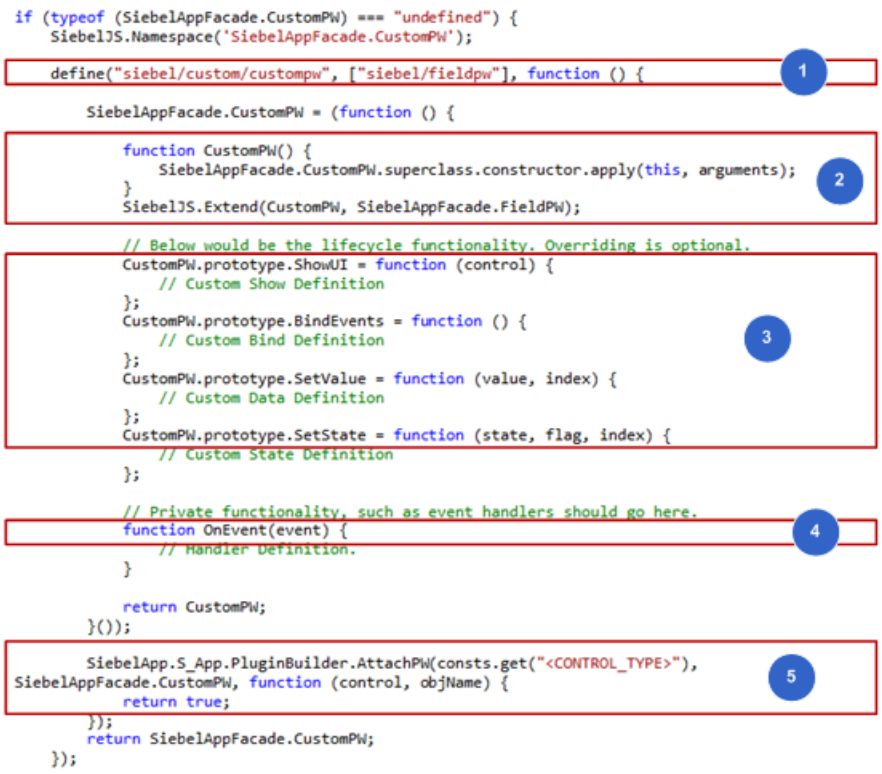
This topic describes the skeleton structure of a plug-in wrapper.

In some older releases, you might have customized Siebel Open UI to control behavior in physical renderers. This type of customization is now done in custom plugin wrappers. The skeleton structure in this topic provides a broad overview of what parts of a plug-in wrapper you might need to customize to achieve parity with your previous customizations.

The following figure illustrates the basic structure of the code you use to override a plug-in wrapper. The example code, would be contained in an independent file in the following directory:

AI_INSTALL_DIR\applicationcontainer_external\siebelwebroot\scripts\siebel\custom

For information about deployment and manifest configuration, see *Example of Customizing Siebel Open UI*.



Explanation of Callouts

As shown in this image, you use the following code structure to override a plug-in wrapper:

1. **Definition of the Custom Plug-In Wrapper.** Defines a custom wrapper by following the same ideology as presentation models and physical renderers. It lists the file of the wrapper and lists the dependencies. This will act as a module identifier to the RequireJS plug-in that Siebel Open UI uses to manage all client side modules.

   > **Note:** Controls that use third-party files list the dependencies here. They must be moved out of the renderer that listed this external dependency.

2. **Constructor.** Does the required changes and calls the superclass. The choice of class from which to extend is decided in this section. It can be a new plug-in wrapper, or subset deviation from an existing plug-in wrapper.
3. **Lifecycle Methods.** Specifies the methods that have been overridden. There is no need to override methods that have not been customized, because the superclass method will be called in instead. Here are the methods that are overridden in this example:
   - **ShowUI.** Override this method to modify display level changes to the control. You can create new DOM or modify an existing DOM that is created by the superclass.
   - **BindEvents.** Override this method to customize event handlers for the control. You can create new handlers in addition to those being added by the superclass or create an entirely different set by not calling the BindEvents in the superclass.
   - **SetValue.** Override this method to affect any value changes to the control. Decorate the exiting value display mechanism or change it completely by avoiding the call to the superclass.
   - **SetState.** Override this method to affect any state changes to the control. Control the behavior when changes to the state; such as editability, focus, and others; are requested. One or more states can be affected by controlling calls to the superclass.
4. **Private Methods.** Use private methods like in presentation models and physical renderers to handle custom functionality, for example: Event Handler definitions.
5. **Declaration of Custom Plug-in Wrappers.** This is the section that declares to the framework that a custom plug-in wrapper has been deployed. It describes the conditions under which the plug-in wrapper needs to be used. Its parameters are:
   - **Control Type.** The SWE constant for the type of control for which the functionality is trying to be overridden.
   - **PW Class.** The class name of the custom plug-in wrapper.
   - **Return Conditions.** This describes the conditions under which the extended plug-in wrapper should be used. A return of `true` will attach the extension.
   - **Control Object.** The instance of the control object for which the decision needs to be made. All control object APIs are available here. Use this object and its APIs to evaluate the return value to `true` for the specific control or controls for which the extension needs to be applicable.
   - **objName.** The name of the object, applet or view for which the decision needs to be made. Use this to evaluate the return value to `true` for the specific applet for which the extension needs to be applicable.

About Presentation Model-Injected APIs in Plug-in Wrappers

This topic includes information about presentation model-injected APIs in plug-in wrappers.

You can use the APIs in this topic to achieve the customized functionality you were using previously in renderers. The code examples for each of the methods in this topic can be used as references about achieving the same or similar customizations that you might have had in some older releases.

For more information about plug-in wrappers, examples of configuration and further information about APIs, see *Architecture of Siebel Open UI*, *Example of Customizing Siebel Open UI*, and *Application Programming Interface*

While the following APIs have been described earlier in this documentation in the context of use within the physical renderer to act as liaisons with the plug-in wrapper, they are also available for use from within the plug-in wrapper:

- **GetEl.** Used to get the DOM (jQuery element) that represents the control.
- **GetValue.** Used to access the DOM value of the control.
- **SetValue.** Used to set the DOM value of the control.
- **SetState.** Used to set various DOM states for the control.

In addition to the APIs listed here, the framework injects certain presentation model-level APIs into all plug-in wrappers to ease the layering of calls that a plug-in wrapper is required to complete. This makes programming in customized plug-in wrappers very similar to programming customized physical renderers. These types of APIs are described in the following topics.

Get

This API gets the value of a PM property where the control is deployed.

SetProperty

This API allows the plug-in wrapper set a property on the PM on which this control is operating. They are PM properties that the plug-in wrapper is acting on.

The following example shows how a CustomPW can operate on a PM property, and how another CustomPW can subsequently use the property for other purposes:

```
CustomPW1.prototype.ShowUI = function (control) {
// Custom Show Definition
if (this.Get("ShowControl1") === true) {
SiebelAppFacade.CustomPW1.superclass.BindEvents.call(this);
}
this.SetProperty("ShowControl2", false);
}
 CustomPW2.prototype.ShowUI = function (control) {
// Custom Show Definition
if (this.Get("ShowControl2") === true) {
SiebelAppFacade.CustomPW2.superclass.BindEvents.call(this);
}
}
```

ExecuteMethod

This API is similar to the ExecuteMethod in the presentation model: it allows the plug-in wrapper to execute a method on the PM on which it is operating.

OnControlEvent

This API runs the event handler call up to the presentation model and subsequently any custom event handlers that may be attached to the event.

The following example shows the usage of APIs in a custom event handler on a custom plug-in wrapper. It depicts a method that is executed on the presentation model, the return value determines if the event should be handled or ignored:

```
function OnClick(evt) {
// This is a custom click handler for my control.
var self = evt.data.ctx,
shouldHonorClick = self.ExecuteMethod("CustomPMMethod");
if (shouldHonorClick) {
 self.OnControlEvent("customClickEvent", self.control, self.dataset);
}
}
```

Helper

This API is used to obtain helper objects from the plug-in wrapper. Currently only the EventHelper object is present. Any control level custom binding must happen using this helper object.

The following example binds two events on to the control in the customized wrapper, and runs custom handlers defined in the wrapper:

```
CustomPW.prototype.BindEvents = function (control) {
var ele = this.GetEl(),
evHelper = this.Helper("EventHelper");
evHelper
.Manage(ele, "click", { ctx: this }, OnClick)
.Manage(ele.next("span"), "hover", { ctx: this }, OpenAlertBox);
};
function OnClick() {
// Custom Definition for element click here
}
function OpenAlertBox() {
// Custom Definition for element next span hover here
}
```

For a detailed example about using APIs in a customized wrapper, see *Configuring the Manifest for the Color Box Example*.