Configuring Siebel eBusiness Applications

Version 7.7, Rev. A
September 2004

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404

Contents

Chapter 3: Working with the Entity Relationship Designer

Chapter 4: Configuring Tables and Columns

Chapter 5: Configuring EIM Interfaces

Chapter 6: Configuring Docking Rules

Chapter 7: Configuring Business Components

Chapter 8: Configuring Joins

Chapter 9: Configuring Links

Chapter 13: Configuring Screens and Views

Chapter 14: Configuring Applications

Chapter 15: Configuring Web Page Objects

Chapter 16: Configuring Toolbars and Menus

Chapter 17: Configuring Picklists and Pick Applets

Chapter 18: Creating and Administering Lists of Values

Chapter 19: Configuring Multi-Value Group and Association Applets

Chapter 20: Configuring Special Purpose Applets

Chapter 21: Configuring Special Purpose Controls

Chapter 22: Displaying Images

Chapter 23: Overview of Web Templates and Siebel Tags

Chapter 24: Specialized Behavior Supported by Web Templates

Chapter 25: Configuring Keyboard Accelerators

Chapter 26: Configuring Spell Check

Chapter 27: Configuring the Customer Dashboard

Chapter 28: Online Help Development

Index

1 What's New in This Release

Configuring Siebel eBusiness Applications is a new book title for 7.7 that covers how to configure Siebel applications. For example, it describes the Siebel object architecture, how to extend the data model, define business logic, and configure user interface objects. It does not cover the Siebel Tools user interface or general tasks about using Siebel Tools, such as customizing the Siebel Tools environment, checking in and checking out, and compiling. For information about the Tools application, see *Using Siebel Tools*.

The content covered in *Configuring Siebel eBusiness Applications* came from the following documents:

- Most conceptual information and information about how to configure Siebel objects came from *Siebel Tools Reference*.

  Siebel Tools Reference is no longer published. It has been replaced by *Using Siebel Tools* and *Configuring Siebel eBusiness Applications*.

- Guidelines for configuring Siebel objects came from *Configuration Guidelines*.

  Configuration Guidelines is no longer published.

- Information about developing online help came from *Siebel Developer's Reference*.

What's New in *Configuring Siebel eBusiness Applications*, Version 7.7, Rev. A.

Table 1 lists changes described in this version of the documentation.

Table 1. Changes in Configuring Siebel eBusiness Applications, Version 7.7, Rev. A

Topic	Description
"Browser-Side Scripting" on page 63	Modified steps for loading compiled browser scripts into the Siebel Web server extension.
"Process for Creating Screen Home Page Views" on page 308	Added process for configuring screen home page views.
"About the Toolbar Item Object Type" on page 330	Added new description of the Position property for the Toolbar Item object type.
"About the Menu and Menu Item Object Types" on page 330	Added new description of the Position property for the Menu Item object type.
"About Lists of Values" on page 372	Revised description of Lists of Values to include an example and information on how LOV Types map to Picklist objects in the repository.

Table 1. Changes in Configuring Siebel eBusiness Applications, Version 7.7, Rev. A

Topic	Description
"Creating New LOV Types and LOV Values Using Siebel Tools" on page 372	Revised procedure to include information about defining records to define LOV Types.
"About Organization Enabled Lists of Values" on page 374	Added section to describe new 7.7 functionality for configuring LOVs to appear for specified organizations.
"About Shuttle Applets" on page 417	Removed note about using CFG file parameters to disable the shuttle applet functionality. Added note about shuttle applets not supporting popup applets.
"Creating a Spell Check Button" on page 554	Revised procedure to cover differences when configuring spell check for required versus non-required fields.
"Creating a Spell Check Menu Item" on page 555	Revised procedure to the differences between configuring spell check for required versus non-required fields.

What's New in *Configuring Siebel eBusiness Applications*, Version 7.7

Table 2 lists changes described in this version of the documentation to support release 7.7 of the software.

Table 2. New Product Features in Configuring Siebel eBusiness Applications, Version 7.7

Topic	Description
"Working with the Entity Relationship Designer" on page 75	New chapter about the Entity Relationship Designer.
"Changing Styles of Label Text" on page 257	Added this topic about using HTML tags in object properties.
"Aligning Label Text in a Grid Layout" on page 270	Added this topic about aligning label text independently from controls in grid-layout applets.
"Copying and Pasting Items in a Grid Layout" on page 271	Added this topic about copying and pasting items.
"Applet Web Templates that Cannot Be Converted to a Grid Layout" on page 276	Revised this topic for new 7.7 functionality.
"About the UI Navigation Model" on page 279	Revised this topic to support new 7.7 functionality.
"Registering Views" on page 287	Revised this topic to include information about the new 7.7 View picklist.

Table 2. New Product Features in Configuring Siebel eBusiness Applications, Version 7.7

Topic	Description
"About Screen Views" on page 300	Added this topic to cover new 7.7 Screen View functionality.
"Example Screen View Hierarchy" on page 303	Added this topic to cover new 7.7 Screen View functionality.
"Creating Screen Views" on page 304	Revised this topic to cover new 7.7 Screen View functionality.
"Defining Sequence for Screen View Objects" on page 306	Added this topic to cover new 7.7 Screen View functionality.
"Association Applets Invoked from Multi-Value Group Applets in SI Mode" on page 415	Revised this topic to include information about shuttle applets.

2 Overview of Configuring Siebel Applications

Topics in This Chapter

About Siebel Objects

Siebel *object definitions* are the metadata that define Siebel applications. Siebel object definitions implement the user interface, business entities, and database organization. Object definitions are stored in a set of database tables called the Siebel Repository. Examples of types of objects are applets, views, business components, and tables. You work with object definitions using Siebel Tools.

Object definitions consist of *properties*, which are characteristics of the software construct that the object implements. For example, the properties of a database column include its name, data type, and length.

Object definitions have hierarchical relationships called *parent-child relationships*. For example, when you expand an object, such as Applet, in the Siebel Tools Object Explorer you see child objects, including Applet Method Menu Item, Applet Browser Script, Applet Server Script, and Applet Toggle.

Parent-child relationships between object definitions imply that the child object definition is in, or belongs to the parent object definition, but it does not imply inheritance among object definitions. The set of properties of an object is unrelated to the set of properties of a child object definition.

For more information about Siebel Tools Object Explorer and the Object List Editor, see *Using Siebel Tools*.

NOTE: Terms such as object, property, or class refer to the Siebel application metadata and not to the corresponding terms in object-oriented programming.

About the Siebel Object Architecture

The metadata that defines Siebel applications, objects, and other files such as Web templates and style sheets, can be divided into the following architectural layers:

- **The Physical User Interface layer.** Consists of the physical files, templates, style sheets, and other file-based metadata that render the UI. See "Physical User Interface Layer" on page 27.
- **The Logical User Interface Objects layer.** Consists of user interface object definitions that define the visual interface that the user sees and interacts with in a Web browser. User interface objects are based on business objects. See "Logical User Interface Objects Layer" on page 27.
- **The Business Objects layer.** Objects that define business logic and organize data from underlying tables into logical units. See "Business Objects Layer" on page 29.
- **The Data Objects layer**. Consists of data object definitions that directly map the data structures from the underlying relational database into Siebel applications, providing access to those structures by object definitions in the Business Objects layer. See "Data Objects Layer" on page 31.

The four-layer architecture is shown in Figure 1.

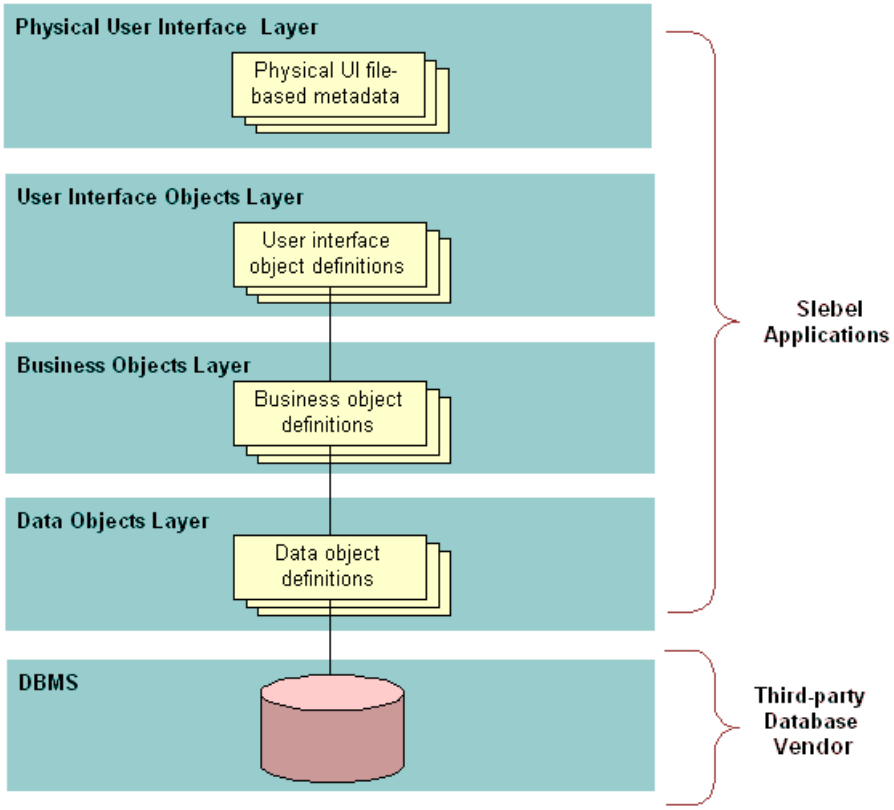


Figure 1. Architectural Layers of Object Definitions

Objects in each layer of the Siebel object architecture are insulated from the layers above it and below it, including the DBMS at the bottom of the architecture. This allows you to change or extend Siebel objects in one layer without impacting other layers. Similarly, because the database in the DBMS is separated from the object architecture, you can make database changes with only minimal impact to the applications.

Physical User Interface Layer

The physical user interface layer consists of physical files, such as templates, Siebel tags, style sheets, and other file-based metadata, that control the layout and the look and feel of the user interface. The physical user interface layer separates style and layout definitions from user interface objects in the repository. This allows you to propagate style and layout definitions across multiple user interface objects.

Table 3 lists physical user interface files.

Table 3. Physical User Interface Files

Files	Description	See Also
Templates files (.swt)	A Siebel template is a special kind of HTML file that defines the layout and formatting of elements of the user interface (such as views, applets, and controls). It provides this layout information to the Siebel Web Engine when rendering Siebel objects in the repository to HTML files.	"About Siebel Templates" on page 483
Siebel Tags	Siebel tags are special tags used in template files. They specify how Siebel objects defined in the repository should be laid out and formatted in the HTML page rendered in the user's Web browser.	"About Siebel Tags" on page 519
Cascading Style Sheets (.css)	Cascading Style Sheets are style sheet documents (of type text/CSS) that define how HTML or XML elements and their contents should appear in a Web document.	"About Cascading Style Sheets" on page 548

For more detailed information about templates, tags, or cascading style sheets, see *Siebel Developer's Reference*.

Logical User Interface Objects Layer

The Logical User Interface Objects layer consists of object definitions that determine the visual interface that the user interacts with in a Web browser. User interface objects expose data from the business object layer and provide users with controls that allow them to navigate, view, and modify data.

Table 4 lists types of user interface objects. For a comprehensive list of all objects and properties, see *Object Types Reference*.

Table 4. User Interface Objects

UI Object	Description	See Also
Applets	An applet occupies a section of a view. Applets allow users to view, enter, modify, and navigate through records. The most common applets appear as data entry forms or lists of records. Applets can also display business graphics, provide a navigation tree, or appear as pop-up windows. Applets are composed of controls, such as buttons and fields, that allow users to interact with data. Applets are associated with data from a single business component. **NOTE:** Applet objects in Siebel applications are not equivalent to Java applets.	"Configuring Applets" on page 231
Views	A view is a collection of related applets that are displayed on screen at the same time. A view is associated with the data and relationships in a single business object.	"About Views" on page 281
Screen	A screen is a collection of related views. Screens are associated with major functional areas of the enterprise, such as Accounts, Contacts, and Opportunities. In general, all views in a screen map to the same business object.	"About Screens" on page 299
Application	An application is a collection of screens. Users access Siebel applications through various clients, including the Siebel Web Client, Mobile Web Client, Handheld Client, or Wireless Web Client. Users navigate to screens from the tab bar and the Site Map, as defined in the application. Your organization may have licensed more than one Siebel application (for example, Siebel Sales and Siebel Call Center), to be used by different groups (for example, the sales team and the customer support team) within your organization. **NOTE:** The Siebel application object is not equivalent to application executable (that is, .exe files).	"About Applications" on page 319
Page Tab	A page tab object is a child of Screen object. It associates a screen to the page tab's parent application object definition and includes it as a tab in the tab bar.	"Defining Page Tabs" on page 321
Screen Menu Item	A screen menu item is a child object of Screen. It associates a screen to the application and includes the screen as a menu item in the Site Map.	"Defining Screen Menu Items" on page 322

Table 4. User Interface Objects

UI Object	Description	See Also
Control	Controls are child objects of Applets. They are user interface elements such as fields, text boxes, check boxes, and buttons. They allow users to interact with the application and with data.	"About Applet Controls and List Columns" on page 238
List	List is a child object type of Applet. A list object defines property values that pertain to a scrolling list table used in list applets and provides a parent object definition for a set of list columns.	"About List Applets" on page 235
List Column	List columns are child objects of List. List columns correspond to columns in a scrolling list table in a list applet, and to fields in the business component.	"Creating List Applets" on page 249
Web Template, Applet Web Template, View Web Template	Identifies external HTML (or other markup language) files that define the layout and Siebel Web Engine interactions for an applet or view.	"About Applets" on page 231 "About Views" on page 281 "About Siebel Templates" on page 483
Applet Web Template Item	Defines a control, list item, or special Web control in the Web implementation of an applet.	"About Applets" on page 231
View Web Template Item	Defines the inclusion of an applet in the Web implementation of a view.	"About Views" on page 281

Business Objects Layer

The Business Objects layer consists of objects that define business logic and organize data from underlying tables into logical units.

Table 5 lists the more common objects in the business object layer. For a comprehensive list of all objects and properties, see *Object Types Reference*.

Table 5. Business Objects

Business Object	Description	See Also
Business Component	A business component is a logical entity that associates columns from one or more tables into a single structure. Business components provide a layer of wrapping over tables, allowing applets to reference business components rather than the underlying tables.	"About Business Components" on page 167
Field	A field object definition typically associates a column to a business component. Fields can also be calculated from the values in other fields in the business component. Fields supply data to controls and list columns in the Web interface.	"About Fields" on page 178
Join	A join object definition creates a relationship between a business component and a table that is not the business component's base table. The join allows the business component to use fields using columns from the joined table. The join uses a foreign key in the business component to obtain rows on a one-to-one basis from the joined table, even though the two do not necessarily have a one-to-one relationship.	"About Joins" on page 199
Join Specification	A join specification is a child object type of join that provides details about how the join is implemented within the business component.	"About Joins" on page 199
Business Object	A business object groups related business components together. Business objects allow you to establish relationships among the business components in the context of the given business object.	"About Business Objects" on page 223
Business Object Component	A business object component is used to include a business component and, generally, a link in the parent business object. The link specifies how the business component is related to another business component within the context of the business object.	"About Business Objects" on page 223

Table 5. Business Objects

Business Object	Description	See Also
Link	A link implements a one-to-many relationship between business components. The Link object type makes master-detail views possible. A master-detail view displays one record of the master business component with many detail business component records corresponding to the master. A pair of links also may be used to implement a many-to-many relationship.	"About Links" on page 206
Multi-Value Link	A multi-value link is used in the implementation of a multi-value group. A multi-value group is a user-maintainable list of detail records associated with a master record. The user invokes the list of detail records from the master record when that record is displayed in a list or form applet. For example, in an applet displaying the Account business component, the user can click the Select button to the right of the Address text box to see a pop-up window displaying multiple Address records associated with the currently displayed account.	"About Multi-Value Links" on page 211
User Property	User properties are object definitions that allow you to configure specialized behavior beyond what is configured in the parent object definition's properties. At the business object layer, user properties exist as child objects of business components, business services, integration components, integration objects, virtual business components.	*Siebel Developer's Reference*
Business Service	A business service is a reusable module containing a set of methods. It provides the ability to call its C++ or script methods from customer-defined scripts and object interface logic, through the invoke-method mechanism.	*Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*

Data Objects Layer

Object definitions in the Data Objects layer provide a logical representation of the underlying physical database (constructs like table, column, and index), and are independent of the installed DBMS. This creates a layer of abstraction over the DBMS, insulating the application and the developer from database administration and restructuring.

Relational database management in Siebel applications is implemented through the layers of Siebel object definitions. The Siebel applications generate queries in response to user actions, in combination with the context established by relevant object definitions. The relational DBMS holds the data, and processes the queries originating in the Siebel application. Query results that are returned from the DBMS are processed up through the relevant object definitions in the Siebel applications architecture, and presented to the user.

NOTE: The physical tables in the DBMS are created as part of the Siebel application installation process.

Table 6 lists the objects in the data layer of the Siebel object architecture. For a comprehensive list of all objects and properties, see *Object Types Reference*.

Table 6. Data Objects

Data Objects	Description	See Also
Table	A table object definition is the direct representation of a database table in a DBMS. It has column and index child object definitions that represent the table's columns and indexes. Table, column, and index object definitions within Siebel Tools provide a detailed picture of all of the tables, columns, and indexes in use in the DBMS.	"About Tables" on page 89
Column	A column object definition represents one column in the database table. Database columns in a database table are represented by the column object definitions that are children of the corresponding table object definition. Each column in the table has a corresponding column object definition.	"About Columns" on page 103
Index	Each index object definition identifies a physical index file in the DBMS.	"About Indexes and Index Columns" on page 107

Summary of Object Types and Relationships

Objects in each layer are built on objects in the layers below. For example, applets are based on business components, views are based on business objects, fields are based on columns. Figure 2 shows the relationships among some of the major object types in a Siebel application.

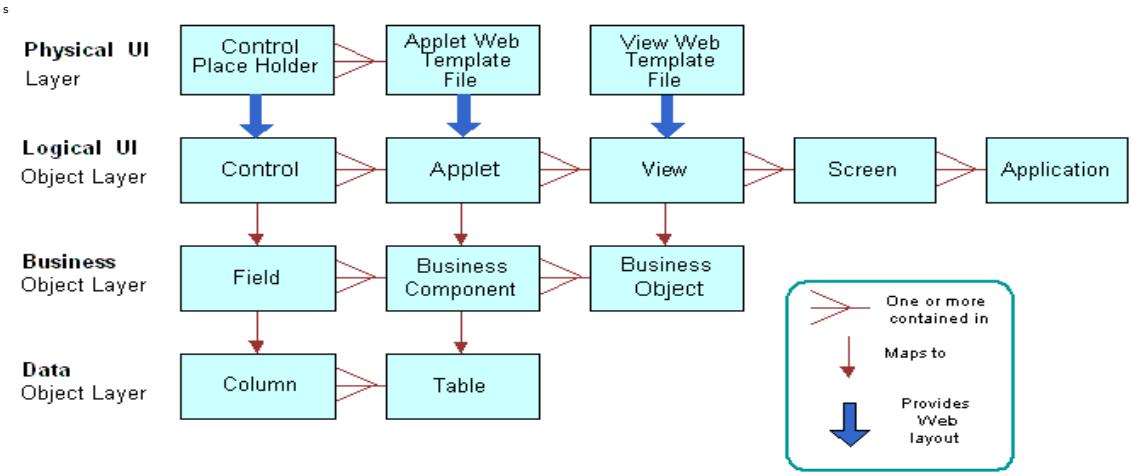


Figure 2. Overview of the Major Object Types and Their Relationships

About Classes in Siebel Tools

In Siebel Tools, *Class* is a property of certain object types, such as applet and business component. The Class property value, assigns a set of behaviors to the object definition, and distinguishes it from other categories of object definitions of the given object type. For example, a value of CSSFrameList in the Class property in an applet object definition makes it a list applet. The Class property accomplishes this by assigning a DLL to the object definition (indirectly by way of a class object definition).

About the Siebel Operating Architecture

Siebel applications are implemented on one or more servers using three major components: the object manager, the Siebel Web Engine, and the data manager. These components are described as follows:

- **Object Manager.** The object manager hosts a Siebel application, providing the central processing for HTTP transactions, database data, and metadata (object definitions in the repository that are relevant to the current state of the Siebel application). The Siebel Web Engine and data manager operate as facilities inside the object manager.

  Object definitions at all three levels of the object layer hierarchy—Web interface definitions, business object definitions, and data object definitions—are processed in the object manager. However, in terms of the run-time objects based on the object definitions, only the business object layer objects (business object, business component, and so on) are instantiated there directly. Web interface objects are instantiated in the Siebel Web Engine, and data objects are instantiated in the data manager.

  The object manager also implements the mechanism by which the Web interface objects receive notification of various state changes of the business component.

- **Siebel Web Engine (SWE).** Generates the user interface in Siebel applications as HTML pages on the server and passes them to a Web browser through HTTP. SWE allows users to view and modify data. SWE retrieves and updates data by interfacing with the object manager. A notification mechanism between SWE and the object manager is used so that when one applet modifies data in any business component, all other applets are notified immediately so that they can update their data on the screen.

- **Data Manager.** The data manager is a facility inside the object manager that issues SQL queries in response to object manager requests, and passes back database result sets to the object manager. The data manager is composed of one connector DLL for each type of database connection supported by the system. The object manager dynamically loads the appropriate DLL based upon the required data source

Figure 3 shows the major entities that make up a Siebel deployment.

Figure 3. Architecture Overview of a Siebel Deployment

For more information about the Siebel environment, see *Siebel Installation Guide* for the operating system you are using and *Siebel Web Client Administration Guide*.

Siebel Web Engine Infrastructure

The Siebel Web Engine (SWE) makes possible the deployment of applications in HTML and other markup languages. A Web client (or other Siebel client) interacts with the server-based object manager through the Siebel Web Engine, as shown in Figure 5 on page 39.

When accessing Siebel applications using the Web client, no components are hosted on the client. The client interacts through a Web browser. The user accesses a specified URL that navigates to a Web-server hosted application. This Web server application is, in turn, supplied with HTML (or equivalent) pages generated by the Siebel Web Engine service in the object manager. The Siebel Web Engine consists of components on two servers—the Siebel Plug-In (also called Siebel Web Extension) on the Web server, and the Siebel Web Engine service in the object manager on the Siebel Server.

A Siebel plug-in (for Microsoft Web server software) runs on the Web server, and interfaces with the Siebel Web Engine service in the object manager. Most of the work takes place in the Siebel Web Engine (SWE); the Web server plug-in mostly maintains the session and functions as a communication intermediary. Network communication between the Web server plug-in and the object manager is through SISNAPI, a TCP/IP-based Siebel Communication protocol that provides a security and compression mechanism.

The Siebel Web Engine runs as an object manager service called the Web Engine Interface Service. This service implements most components of the Siebel Web Engine, deploying an interface between the Siebel plug-in on the Web server and the object manager. From the perspective of the Siebel plug-in, the SWE interface service provides processing for incoming HTTP requests bearing the SWE prefix, and generates HTTP responses. From the object manager's perspective, it provides a user interface in its OM interactions.

Applets and views are made available to the Web by associating a set of HTML templates, which is done using Siebel Tools. At run time, when an applet needs to be rendered, the SWE obtains the information defining the applet, the appropriate data for the various the applet controls or list columns, and the HTML template; it then combines them to generate the final Web page that is then sent to the browser.

Applet Web templates are defined and laid out in Siebel Tools using the Applet Web Template and Applet Web Template Item object types and the Web Applet Designer. View Web templates are defined using the View Web Template and View Web Template Item object types.

How the Siebel Web Engine Generates the Application

Users interact with the application through their Web browsers. The interface they see is a set of Web pages dynamically generated by Siebel Web Engine by matching the repository definition of the application with Siebel Web templates.

When a user interacts with the application, by clicking a button or hyperlink in a browser window for example, the Siebel Web Engine does the following:

1. Reads the object definitions from the .srf file.
2. Selects the necessary web templates.
3. Combines the object definitions and templates.
4. Retrieves data from the underlying business objects and business components.
5. Maps the data and applet and view information to the corresponding placeholders in the .SWT file.
6. Presents the HTML output to the user.

Figure 4 depicts the relationships between style sheets, templates, objects in the repository, and the final HTML output.

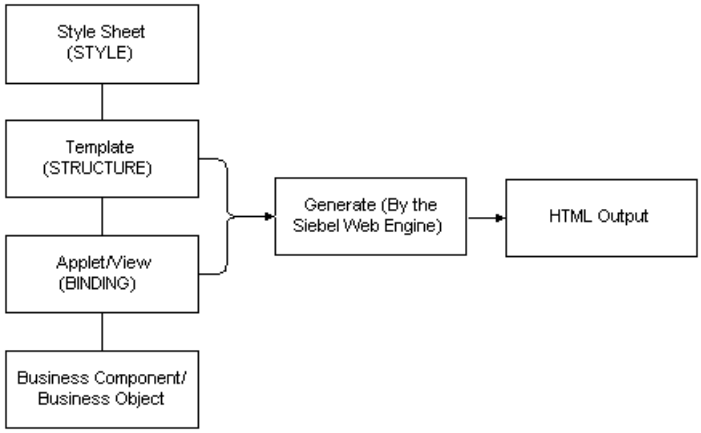


Figure 4. HTML Output

About Standard and High Interactivity

Siebel applications are deployed in either standard interactivity or high interactivity mode.

Standard interactivity mode resembles most traditional Web applications. It supports many different types of browsers. Page refreshes occur often, such as when users create new records, submit forms, and browse through lists of records. Customer applications are deployed in standard interactivity mode.

High interactivity mode is designed to resemble a Windows client. It supports fewer browsers than standard interactivity mode, but it includes a set of features that make data entry easier for users. For example, page refreshes do not occur as often as they do in standard interactivity mode. Users can create new records in a list, save the data, and continue browsing without a page refresh having to occur. Siebel employee applications are typically deployed in high interactivity mode.

Other features supported by high interactivity mode are:

- Browser scripting. In high interactivity, you have access to Siebel objects using browser script. This allows you to build data validation logic on the client side to reduce the number of page refreshes needed.
- Implicit commit. Allows the application to automatically save a record when the user steps off the record.
- User interface features. Features include drag-and-drop column reordering, drag-and-drop file attachments, keyboard shortcuts, pop-up controls for calendar, calculator and currency functions, and applet scroll bars.

- Extensible toolbars. In high interactivity you can extend JavaScript toolbars and create new ones. JavaScript toolbar objects reside in the JSSApplication hidden frame, which usually does not reload during the application life cycle. Therefore, they are not redrawn when there is a page refresh. The UI part of the JavaScript toolbar resides in a visible HTML frame (recommended to be a persistent frame that reloads infrequently) and redraws when the HTML frame reloads.

NOTE: Partner applications can be deployed in either standard interactivity or high interactivity mode.

One way to differentiate between standard and high interactivity modes is by the underlying technologies used by each mode (see Table 7).

Table 7. Technology Differences Between Standard and High Interactivity

Technologies	Standard Interactivity	High Interactivity
Uses Java technology		X
Uses JavaScript technology	X	X
Uses Active X technology		X
Uses Document Object Model		X

JavaScript Object Architecture in High Interactivity

JavaScript is an interpreted language running in many Web browsers. It is used to extend browser behavior. Objects representing the applet, business component, business services, and application object types exist in the browser address space as JavaScript objects and provide communication with the server. These object types are the same object types instantiated within the browser. Initially, these objects are passed through to SWE, but provide caching and other local processing. The JavaScript objects are:

- **Browser Applet.** Provides a framework for communication and interaction between applet controls.
- **Browser Buscomp.** Provides the framework for business component-level interactions. For example, the browser buscomp updates the state of browser applets as values change in the underlying business component (due to parent/child views, calculated values, and specialized behavior).
- **Browser Business Service.** Provides a set of methods from customer-defined browser-side scripts, through the invoke-method mechanism. Browser business services can be reused.
- **Browser Application.** Provides the application-level framework. Methods that are not business component-specific can be accessed here as well as invoke methods on the server.

You can script instances of browser applets, browser buscomps, browser business services, and browser applications. For more information, see "About Scripting and Object Interfaces" on page 62.

In the diagram in Figure 5, the different boxes represent different components or different parts of the application. Specialized business component logic is shared among all platforms; specialized Web applet logic is shared between all HTML clients; and browser logic is the only part that is browser-specific.

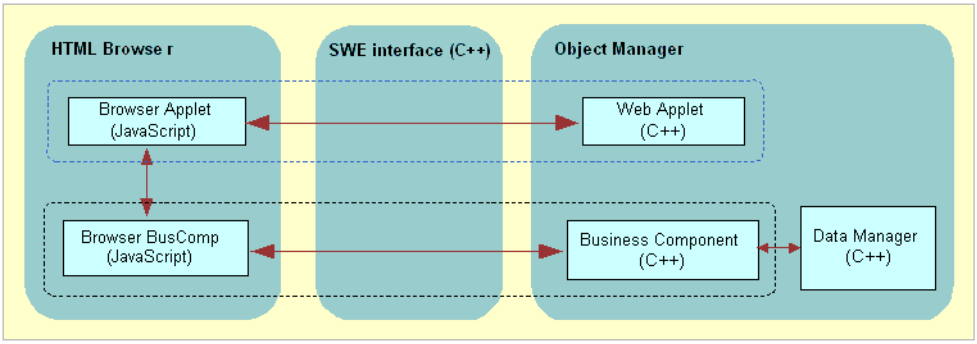


Figure 5. JavaScript Architecture for High Interactivity

These browser-side JavaScript objects are maintained in sync with their server-side counterparts, so that changes on the browser or server objects are reflected in their corresponding objects. Application processing is performed among the browser-side objects. Using the remote procedure call protocol, the server is activated when data or new layouts are required from the server. The server can also initiate actions on the browser, using the notifications protocol.

Enabling and Disabling High Interactivity for Applications

Employee applications are set to use high interactivity mode by default. You can also set employee applications to run in standard interactivity mode. The mode is determined by a parameter in the application configuration file.

NOTE: high interactivity is not supported for customer applications.

To enable or disable high interactivity

1. Verify that the application is an employee application.
2. Open the application configuration (.cfg) file.
3. In the SWE section in the .cfg file, do one of the following:
   - To enable high interactivity, add the parameter *HighInteractivity* with value = *TRUE*:

     ```
     [SWE]
     HighInteractivity=TRUE
     ```

- To disable high interactivity (run standard interactivity mode), delete the HighInteractivity parameter from the .cfg file.

4 Save and close the .cfg file.

NOTE: When running an application with HighInteractivity=TRUE, the Web framework attempts to show views in high interactivity only if every applet contained in the view supports this. Otherwise, the view will be displayed in standard interactivity.

About Configuring Objects for High Interactivity

When configuring applications for deployment in high interactivity mode, consider the following:

- Browser scripting is fully supported in high interactivity mode.
- For fields to interpret and display custom HTML, such as a URL entered by the user, the field's Type property must be set to URL. If it is not set to URL, the HTML is presented and interpreted as plain text. For example, if a user typed a URL in a field of type TEXT, the URL would not be recognized as a link to a Web page.
- You cannot modify the appearance of the rich text editor.
- You cannot modify the background and text color of list applets.
- You cannot place method-invoking controls, such as the delete function, on every row in a list. Instead, place a button on the applet that calls the method. The function will act on the selected record.
- There are cases when an application's configuration file is set to run in high interactivity mode and all the applets in a view are configured to support high interactivity, but the view appears in standard interactivity mode. Reasons this might occur are:
  - One of the applets is in the Query mode. Because high interactivity implicitly supports query operations from the user interface, it does not support the explicit use of the Query mode.
  - One of the applets is in the New mode and uses a New template that is different from the Edit template used in its default mode. This can be avoided by inactivating New templates associated with the applets used in high interactivity applications. The framework will then default to using the Edit template itself to create new records.
  - One of the list applets has multi-row edits or multi-row select enabled.
  - One of the list applets is a hierarchical list applet.
  - The view uses a template that shows applets in a catalog-style layout. None of the employee applications should be using this layout.
  - A combo box picklist uses Long Lists or has an associated pick applet. For example, if you perform an action from a high interactivity applet that causes a pick applet to be displayed, the pick applet will not be in high interactivity mode.

About Calendar Views and Interactivity

Most views in the Siebel applications can display in both the standard interactivity mode and the high interactivity mode. Some calendar views are different: they run in either high interactivity mode or standard interactivity mode, but not both.

For Siebel applications that typically run in high interactivity mode, the seed responsibilities have been set with the high interactivity views and also with those views that can be shared by both high interactivity and standard interactivity modes. For Siebel applications that typically run in standard interactivity mode, the seed responsibilities have been set with the standard interactivity views and with those views that can be shared by both high interactivity and standard interactivity modes, as shown in Table 8.

Table 8. Application Modes for Calendar Views

View Name	High Interactivity Mode	Standard Interactivity Mode Only	Both Modes
High Interactivity Activity Calendar View	X		
eCalendar Daily View		X	
eCalendar Monthly View		X	
eCalendar Weekly View		X	
eGanttChart View			X
eCalendar Detail View			X
eCalendar Detail View With Participants			X
Calendar Access List View			X

If you plan to deploy an application that typically runs in high interactivity mode in standard interactivity mode, you need to do the following:

- Remove any high interactivity-mode calendar views from all user responsibilities.

  For information about modifying responsibilities, see *Security Guide for Siebel eBusiness Applications*.

- Add all standard interactivity-mode calendar views to all user responsibilities.
- Retarget any links to calendar views so that they point to the appropriate standard interactivity-mode calendar views.

On occasions when your site decides to run applications in high interactivity mode for some users and standard interactivity mode simultaneously for other users, you should create two sets of responsibilities: one set for high interactivity mode users and another set for standard interactivity mode users. Then, assign users to the responsibility appropriate for the mode in which they run the application. Under these circumstances, it is recommended that you deactivate links to the calendar views, because the same link cannot point to both a high interactivity view and a standard interactivity view.

About Integration with J2EE

Many enterprises, especially those involved in eBusiness, develop and implement Java applications to meet a variety of business requirements. Typically, these applications combine existing enterprise information systems with new business functions to deliver services to a broad range of users.

Such services are usually architected as distributed applications consisting of three tiers: clients, data sources, and the middle tier between the two. The middle tier is where you typically find transports and interfaces such as HTTP and MQSeries, as well as Java servlets and Enterprise Java Beans (EJBs) to receive the messages (typically, these are in XML format) between applications inside and outside the enterprise.

To further simplify integration, Siebel applications provide a Java/XML Framework designed to receive XML requests sent by Siebel over HTTP or MQSeries. The Java/XML Framework provides a uniform way to receive and process Siebel application requests within J2EE environment. Requests initiated from within Siebel applications are transmitted to the appropriate J2EE Application Server using Siebel's eAI integration infrastructure. The Java/XML Framework consists of a Servlet to receive HTTP requests and an MQSeries Base Server designed to retrieve messages from an MQSeries queue.

When implementing the Java/XML Framework, you will need to implement a single interface (ProcessRequest) responsible for understanding the contents of the incoming request and dispatching it to the appropriate Java component.

CAUTION: The Java/XML Framework may be used only to receive XML requests from the Siebel programs. This code may be extended solely for use in object code form and solely for the purpose of integrating the Siebel programs with non-Siebel applications; however, any modification or extension of this code is outside of the scope of Maintenance Services and will void all applicable warranties.

In addition to the Java/XML Framework described above, you can generate JavaBeans that represent Siebel Integration Objects or Business Services using the Siebel Code Generator Business Service. The JavaBeans generated by the Siebel Code Generator provide a strong interface for Integration Objects, Business Services, and their related components, allowing you to identify and use the Java code you need for your application.

CAUTION: The source code generated by the Siebel Code Generator Business Service may be used only in object code form and solely for the purpose of integrating the Siebel programs with non-Siebel applications. Any modification or extension of code generated by the Siebel Code Generator Business Service is outside of the scope of Maintenance Services and will void all applicable warranties.

For additional information regarding the Java/XML Framework and the Siebel Code Generator Business Service (Java Wizard), see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*.

About Siebel Partner Connect and Siebel Tools for Partner Connect

Siebel Partner Connect is a business-to-business integration solution that allows brand owners to deploy integrated processes with their demand-chain partners.

Siebel Tools for Partner Connect is a set of tools that brand owners use to configure and administer their integrations with their channel partners. It includes the following webMethods products:

- webMethods Developer
- webMethods Trading Networks Console
- webMethods Business Integrator

For more information on Siebel Partner Connect and Siebel Tools for Partner Connect, see *Siebel Partner Relationship Management Administration Guide*.

About Configuring Siebel Applications

Configuration is the process of altering standard Siebel applications to meet business requirements. This can range from making minor changes, such as adding text box controls (and their underlying fields), to creating new user interfaces and business entities.

Siebel Tools is the software application that allows you to reconfigure and extend Siebel applications. It is a software configuration toolset rather than a programming language. What this means is that software is developed and enhanced by creating and modifying object definitions and their properties.

The Siebel applications software is built on object definitions that are executed at run time, and are available to the developer to modify. By creating new object definitions (and adapting existing ones to new uses) you can create complete new modules. It is not necessary for you to write C++ program code, although you may want to write Siebel Visual Basic (VB), eScript, or browser script to supplement the programmatic logic of your application.

NOTE: Siebel VB and eScript runs on the server side; Browser script runs on the client side.

To see other methods of configuration in Siebel applications, see "Other Ways to Customize Application Behavior" on page 68.

Usage and Configuration of Non-Licensed Objects

The licensing agreement between Siebel Systems, Inc. and its customers is such that customers are only entitled to use and configure Siebel objects (for example, business components and tables) that belong to modules they have purchased.

If a Siebel object is not exposed to the licensed user interface—through views that are exposed under the customer's license key—the customer is not entitled to use that object in custom configurations.

Customers are, however, entitled to create new tables using Siebel Database Extensibility features and to create new business components and UI objects to expose these tables.

Configuration Goals and Objectives

The major goal of Siebel application configuration is to create a target application that meets the look, feel, and functional requirements of your organization and your users—and is easy to maintain and upgrade.

Key objectives for your configuration project should include:

- Use existing Siebel application functionality (that is, never create new objects unless your requirements cannot be met by modifying existing ones).

  If you follow this principle your configured application will be much easier to maintain and upgrade to future Siebel-product releases.

- Standardize configuration development.
- Achieve acceptable system performance.

  For information performance, see *Performance Tuning Guide*.

- Build a consistent and intuitive user interface.

  For example, if you create a new form applet it should have the same general look and feel as other form applets in your Siebel application.

Overview of the Development Process

Like other forms of software development, configuring Siebel applications is not a completely serial process. During some phases, it makes sense for multiple pieces to be configured concurrently. Furthermore, some tasks—most obviously testing and debugging—are iterative, more like a loop than a straight line. For this reason, it is likely that you will modify the simplified, rather linear process described in this section to suit the needs of your team.

A simplified summary of a typical application development process might consist of the following:

1. Do a thorough business analysis of your organization's and users' needs, and get approval and time and resource commitments from the relevant organizations.
   - Can you meet the needs of your users with a standard Siebel application?

- If not, what business needs will require changes to the application?
- How can you assure success with your configured application?

2 Write design documents that include:

- The requirements that are being satisfied by the configured application.
- An entity relationship diagram (ERD) or text equivalent of the entity relationships.

  For more information about ERDs, see "Working with the Entity Relationship Designer" on page 75.
- The names and descriptions of the business objects and business components required for your application, and how they relate to one another.
- Screen flow diagrams and a list of fields to be displayed on each applet.
- A description of your development environment and process, for example:
  - How will the work be divided up among participating developers?
  - Naming conventions the development team will be required to use.
  - How will the application be tested and rolled out to users?
- The complete step-by-step procedures your development and test team will need to follow to complete the application.

3 Have the design reviewed by your participating organizations and users.

4 Set up your application development environment—for example:

- System and database environment.
- Developer workstations.

5 Develop the application:

a Use Siebel Tools to modify (or create, if necessary) the necessary object definitions:
  - Data objects such as tables, columns, and indexes.
  - Business components and business objects.
  - User interface objects (for example, applets, views, and screens).

b Modify Web template files.

c Compile your Siebel application and do unit testing.

6 Using the tools available to you in the Siebel application environment (for example, Siebel Assignment Manager and Siebel Business Process Designer), implement the appropriate assignment and workflow rules.

7 If necessary, extend the functionality of your application through scripting (Siebel VB or Siebel eScript).

8 Localize your application if the user interface is to be displayed in two or more languages.

9 Do system and performance testing of your Siebel application.

10 Iterate through the development steps until your design has been fully implemented and your application is running smoothly and meets your performance objectives.

11 Introduce the application to your users and train them.

About Structuring Development Work

There are two common approaches to structuring the development work required to configure Siebel applications:

- Assign a single developer or group the development role for a complete functional area.
  - For example, one group or an individual person may develop a Web page and all the supporting logical business object definitions and data object definitions.
  - This approach typically enables different groups to implement in parallel.
- Assign a single developer or group to a specific architectural layer.
  - This approach takes advantage of the specialized expertise of developers—for example:
    - The RDBMS specialists can implement extensions in the Data Objects Layer.
    - The system architects can implement the Business Object Layer.
    - The UI developers can implement the User Interface Objects Layer.
  - Using a Web template requires each group to complete some work before another group begins.

Configuration Strategy

- Make as few changes as possible.
- Reuse existing objects in the Siebel Repository when the intended purpose of the object is a good fit (technically and functionally) with your business requirements. See "About Object Reuse" on page 51.
- Plan your application design from the top down. Design the user interface, define the underlying business logic, then define the data model necessary to support your configuration.

- Configure from the bottom up. Modify objects at the data layer, then business object layer, and finally configure user interface objects, as shown in Figure 6. This helps you make sure the correct values for all required object properties are available as options.

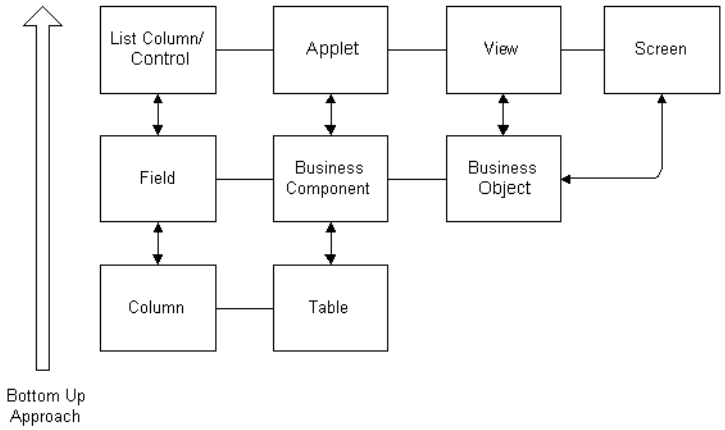


Figure 6. Bottom Up Approach to Configuring Siebel Applications

Guidelines for Naming Objects

When naming objects consider the following:

- Never include a question mark at the end of a field name or user interface label.
- Use meaningful, descriptive object names (for example, Account Detail Applet With Agreement, instead of Account Detail Applet 2).
- Be careful about spelling, spacing, and capitalization when naming objects. Typically, logical names of objects in the repository use complete words, mixed casing, and spaces between words. However, physical database objects use abbreviations, uppercase, and underscores. For example, the Service Request business component is based on the S_SRV_REQ database table.
- It is time-consuming to change an object's name after it has been referenced throughout the repository. If you need to change the name of an object that may have many references throughout the repository, use the Find in Repository feature (from the Tools menu in Siebel Tools) to find all of the references.

TIP: If you are in doubt about object names, use the existing objects in the standard Siebel repository as your guide. For example, when creating a new Association applet, you would notice the <BusComp> Assoc Applet naming convention. Examine the standard objects and conform to their established naming conventions.

For naming conventions for specific object types, see the following topics:

"About Tables" on page 89

"Guidelines for Configuring Business Components" on page 171

"Guidelines for Configuring Joins" on page 202

"Guidelines for Configuring Business Objects" on page 227

"Guidelines for Configuring Applets" on page 236

"Guidelines for Configuring Views" on page 284

About Modifying Objects

When possible, modify existing objects rather than create new ones.

- Many existing objects have been configured for best performance; new or copied objects may not automatically inherit this same ability.
- Because you have an archived copy of the original object to revert back to (in the sample database), troubleshooting will be much easier. You can also use the comparison feature in Siebel Tools to determine what changes were made to the object that might be causing the problem.
- Repository and application maintenance requires less time and fewer resources.
- The SRF is smaller and compiles faster.
- Eliminating unnecessary copies of objects reduces the amount of redundancy in the repository.
- Because standard objects have already been thoroughly tested, less effort is required to test the application or resolve application errors.
- By reducing the number of repository objects being evaluated or upgraded, there is less effort required when upgrading your application.

About Copying Objects

Use caution when copying objects. Although there are cases when copying objects is appropriate, there are also a number of problems associated with copying objects.

Problems Caused by Copying Objects

Copying objects can result in the following problems:

- Copying can create upgrade problems that are difficult to debug.

  Functionality is often added to most of the standard business components during major releases. Very often this new functionality depends on the existence of new fields, joins, and so on, that are added to the standard business components. During the upgrade, these new fields are added only to the standard business components in your merged repository.

  Copying objects can result in problems following an upgrade. These problems can be difficult to locate and debug. The errors often occur because some C++ code for the business component or applet class is trying to find a field that does not exist in your custom copy of that business component or applet. The only way to debug the problem is to compare your custom business component with the standard business component and add any new fields and other child object definitions that may have been added in the new release. This may be a complex process, requiring detailed knowledge of what has changed in the new release.

- Copying objects creates redundancy.

  Creating new copies of business components and applets can result in considerable redundancy in your configuration. For example, if you were to create a copy of the Account business component called My Account, and use this on all of the account-based views, you would also have to create copies of every account-based applet and point each to the new My Account business component. You probably would also have to create a new business object, screen, and so on. It would result in considerable additional configuration with little or no benefit.

- Copying objects can increase complexity.

  Sometimes developers make copies of object definitions in the belief that doing so will reduce problems during an upgrade. The assumption is that if the business component is called My Account, the Application Upgrader will leave it alone during the upgrade, resulting in no problems after the upgrade.

  However, this assumption is misleading. The problems you will have with an upgraded configuration containing copied object definitions will be more complex to solve than the problems potentially caused by reusing standard object definitions. It is far easier to go through your application after an upgrade and remove various new controls and list columns from a standard applet than it is to go through each custom business component and applet and work out what fields, joins, multi-value links, and so on to add.

Copying User Interface Objects

Sometimes it is appropriate to copy user interface objects. For example, when business requirements call for significant changes to the look and feel of the object, copying the object (and setting the Upgrade Ancestor property) makes certain that the modified look and feel would be preserved following an upgrade. If only minor changes to the UI object are needed, it is better to use the out-of-the-box object because this eliminates time spent configuring and the continuing maintenance of the repository. Other appropriate reasons for copying UI objects are:

- When two different UI objects have to display different records (that is, different search specifications on applets).
- When different read/write properties between two objects are necessary (that is, one applet is read-only, and the other is editable) and if this could not be accomplished through the dynamic read-only business component user property.

- When different drilldowns are necessary for different applets, depending on the view that contains them (provided this could not be accomplished through a dynamic drilldown).

When copying an applet that uses a business component based on a specialized class, the following guidelines apply:

- The copied applet *must* be used with the original business component, not a copy of the original business component.
- If you want to use a copied applet with a copied business component, you need to change the class of the copied applet.

Copying Business Objects and Business Components

In general, avoid copying business objects and business components. However, copying may be appropriate when:

- A business component must appear twice in a business object; for example, when the Account business component and Sub Account business component both appear within the Account business object.
- Two business components contain different search specifications *and* predefault values for the Type field that differentiates the records of these two business components.

About Upgrade Inheritance

The Upgrade Ancestor property allows copied objects (of types Applet, Business Component, Report, and Integration Object) to inherit properties of the original objects from which they were copied. During an upgrade, changes applied to the original objects are also applied to copied objects.

For example, suppose you create a copy of the Account List applet, call it the Premium Account List applet, and set the Upgrade Ancestor property. The new applet may differ from the original in that it has a special search specification that is displayed only in accounts that are considered premium accounts. In a subsequent release, Siebel Systems, Inc. may add new out-of-the-box list columns to the Account List applet. During an application upgrade, your Account List applet and the Premium Account List applet will retain the configuration changes you made. However, both applets also receive the new out-of-the-box list columns added in the new version.

When using the Upgrade Ancestor property, consider the following:

- Use caution when copying objects. For more information, see "About Copying Objects" on page 48.
- The upgrade ancestor property is not automatically defined when you copy an object. You must define it manually.
- Creating a new object without specifying an Upgrade Ancestor property could add to your upgrade efforts, as custom objects will *not* be upgraded. Instead, they are copied to the new repository, but without changes.
- Creating new copies of business components and applets can create a significant amount of redundant configuration.

For more information about upgrade inheritance, see *Upgrade Guide* for the operating system you are using.

About Object Reuse

In general, reusing existing objects is the recommended approach for configuring a Siebel application to your meet business requirements. However, there are cases when reusing existing objects is not appropriate and can cause problems. This section provides guidelines about when to reuse and when not to reuse existing Siebel objects.

Customizing Siebel Applications

Avoid significantly customizing Siebel applications. Customization refers to large-scale changes to the base product, such as:

- Creation of new modules that do not exist within the Siebel application, usually involving use of database extensibility, many new business components, and many new business objects.
- Significant modification to existing objects.
- Significant changes to out-of-the-box behavior, such as visibility, and changes to framework objects such as JavaScript files.
- Use of scripting.

Inappropriate customization of the Siebel application can cause the following issues:

- Decreased maintainability and therefore increased cost of ownership.
- Potential for decreased in performance.
- Potential impacts with future upgrades.
- Increased testing effort.

Developers should avoid significant customization of the Siebel application, and attempt to reuse and extend existing objects where possible. However, there are cases where reuse is inappropriate.

Why Reusing Objects is Important

Reuse in software development involves the building of components that can be reused and extended. Reuse allows you to limit the amount of customization in your deployment. Any changes to the out-of-the-box configurations should maximize reuse and allow for easy extensibility.

Reuse is important for the following reasons:

- It optimizes the performance of Siebel applications. Out-of-the-box Siebel configurations have been tuned for performance.
- Provides for easier maintainability and reduced cost of ownership.
- Facilitates future upgrades.
- Consistency in application behavior.

There are several ways to reuse components in Siebel applications, for example:

- Use Siebel out-of-the-box functionality and modules without substantial modification.
- Use out-of-the-box configuration objects such as business components, business objects, links, applets, views and so on.

  NOTE: In general, avoid copying objects. See "About Copying Objects" on page 48.

- Use declarative techniques to translate business requirements into application behavior, such as configuration in Siebel Tools, Workflow, Personalization, Runtime Events, and State Model.

Reusing Objects Inappropriately

While reusing Siebel objects is generally the recommended approach for configuring Siebel applications to your business requirements, reusing Siebel modules or repository objects in ways that do not suit their original purpose can cause a number of problems.

- The performance of the component or object has been tuned by Siebel Engineers with a specific purpose in mind. Using it for a different purpose may result in performance degradation.
- Using a repository object for the incorrect purpose may limit your ability to use that object for its intended purpose in the future. For example, you may choose to use the Service Request business component for a custom purpose. Doing so limits your ability to use this business component for its correct purpose in the future, which is to store a service interaction with a customer.
- Large amounts of customization may be required to make the improper use of an object function. For example, when reusing a table for an alternative purpose, it is necessary to populate all required columns as well as add a search specification to the business component. If a business component is used for an improper purpose, there may be specialized class behavior that affects your ability to properly use the business component. Additional customization may be required to make the business component function so that it meets your requirements.
- Inappropriate reuse violates a principle of software development in that the purpose of components should be clear. When an object is improperly used, then the developer's intent is not clear to future developers. For example, if the table S_SRV_REQ is used to store financial transactions, the configuration is not clear to future developers as the table bears no relationship to financial transactions.
- In general, reuse of Siebel repository objects should increase the upgradability of the repository. However, inappropriate reuse can decrease upgradability. Changes to objects during repository upgrades are based on the objects being used for their original purpose. The upgrade may change the table schema by adding or changing unique indexes as well as required columns. It may also change the behavior of specialized classes. Also, Siebel functionality such as access control, visibility, and the party data model may also be changed. Improper reuse of components may cause difficulties during upgrade, which in some cases may be difficult to debug and fix.
- Misusing objects can result in reconfiguration work after an upgrade because the upgrade will revert the object to its original form.
- You can misuse repository objects by reusing objects that are outside of your licensed configuration.

Related Topics

"Deciding When to Reuse Objects" on page 53

"About Reusing Business Component Fields and Table Columns" on page 55

"About Reusing Business Components" on page 58

"About Reusing Tables" on page 59

"About Reusing Views" on page 61

"About Reusing Applets" on page 61

Deciding When to Reuse Objects

In general, use the out-of-the-box objects whenever possible. However, there will be cases when it will be difficult to determine whether to reuse an existing component or to create a new one. This will happen when you have requirements that do not fit within the components that are provided with Siebel out-of-the-box functionality and configuration.

When deciding whether to use an existing Siebel object or to create a new one, you need to determine how suitable the object is for its intended use. The developer should examine both the functional and technical fit of the proposed object. Where the fit is appropriate, then that object should be used. Where it is not, there may be a case for the creation of a new object.

An object should not be reused merely because it is not used by existing Siebel configurations or currently used by the customer. The object needs to also be suitable.

If no object is suitable, you should consider using database extensibility to extend the Siebel application. For more information about extending the database, see "About Database Extension Options" on page 109.

Determining Functional Fit

To determine if an object is a functional fit to your business requirement, you typically examine the table or business component you intend to use and consider the following criteria:

- The nature and purpose of the Siebel object in its native state should be compatible with your proposed usage. For example, storing Customer Complaints is compatible with the Service Request business component, but not for storing Financial Transactions.
- The relationships to other objects and their compatibility with your requirements. However, this should not be the only determining factor. The fact that an object contains the correct relationships is not enough to suggest it is a good candidate for reuse. For example, you should not use the S_EVT_ACT table as an intersection table simply because it contains two of the foreign keys that you require. Doing so causes an overloading of the table that may result in performance degradation.
- The visibility properties of the object and whether they are compatible with your requirements.

If the object is not a good functional fit, then reusing the object for that purpose may be inappropriate. Some examples of improper use include:

- Using the S_PARTY table for storing non-party entities.

- Using unused tables for custom business components where the table has no relationship to the intended usage of the business component.
- Use of unused table columns or business component fields that have no relationship to the intended usage of the field.

One way of improving the functional fit of a Siebel component or object is to modify your business processes to suit out-of-the-box Siebel functionality.

Determining Technical Fit

To determine if an object is a technical fit to your business requirement, examine technical factors, such as:

- Table schema
- Foreign key relationships
- Effect on Siebel Remote
- Performance Factors
- Size and type of columns and fields

When deciding to reuse an unused Siebel table, you need to examine the schema. Some factors include:

- The required columns that need to be defaulted to a particular value.
- The user key and unique index columns that also need to be populated.

If large amounts of customization need to be performed to be able to use an unused table, then technical fit diminishes.

Also, does the table contain the necessary foreign key relationships for your requirements? Again, this is not a sole-determining factor as you should not be simply reusing a table just because it contains the necessary foreign key relationships. It is also possible that Database Extensibility can be used to add the necessary foreign key columns and the Docking and EIM Wizards used if required.

NOTE: Foreign key relationships and Siebel Remote are closely interrelated. Simply using the correct foreign key may not guarantee that the data will be downloaded to the remote client. It is necessary to understand the dock objects and its rules that the foreign keys are subjected to in order to determine if use of the object is appropriate.

Many columns that are not foreign keys have some impact on visibility. For example, S_PROD_INT.ENTERPRISE_FLG with a value of 'Y' confers partial docking visibility to the product record. Misusing columns such as these can have a significant impact on Siebel Remote.

About Reusing Business Component Fields and Table Columns

In general, you should reuse an existing suitable field or column rather than create new ones. However, when deciding whether to use a previously unused field on a business component or an unused column on a table, you need to decide whether that field or column is a good functional or technical fit for your business requirements.

Examples (good and poor) of reusing columns and fields are:

- You can use the Last Name or First Name fields on the Contact business component to store the names of the contact.
- You could use the unused WEIGHT column on S_CONTACT to store the weight of the contact. It should not be used to store other types of information unrelated to the weight of the contact.
- On the Campaign business component, you should not use the field Route Prospects to store information other than indicating that campaign contacts should be routed to the remote client. Storing other information can compromise Siebel Remote performance.
- On the Order Entry business component, you should not use the field Revision to store anything other than the revision number. This field is controlled by specialized behavior.
- On the Account business component, instead of creating a custom extension column for a new field to store an email address, use the unused column S_ORG_EXT.EMAIL_ADDR.

NOTE: The Comment property of the column object type often describes the column's intended purpose. Use comments to help you decide when to reuse a column.

Reusing fields or columns when there is not a good technical or functional fit can have poor results. In these cases, it would be better to create a new field and/or column. Examples of creating and reusing different types of columns and fields are:

- **Duplicate columns**. When deciding to reuse an unused column for a new field on a business component, you need to verify whether the column is not already being used by another field on the same business component. If more than one field maps to the same table column, you may encounter "duplicate column" insert errors during copy operations. In this case, you should try to use the original Siebel field that maps to the desired column if it is suitable. Otherwise, use another appropriate column, such as custom extension column or unused ATTRIB column.
- **Column type and size.** When reusing an unused column, one technical factor is its type and size. Developers are not normally allowed to change either the type or size of the column. An exception is DB2 UDB for OS/390 and z/OS platforms, since the maximum size of VARCHAR columns is stored in indexes.

  For more information, see *Implementing Siebel eBusiness Applications on DB2 UDB for z/OS and OS/390*.

- **Fields use by Siebel Remote.** Do not misuse fields or columns that control Siebel Remote behavior. Usually, the column ENTERPRISE_FLG is used to implement enterprise visibility on records that have normal visibility constraints. Also, other columns may also be used to control routing behavior, such as the RTE_CONTACTS_FLG or RTE_PRSP_CONS_FLG column on S_SRC.

  You can determine if a column is subject to a dock object visibility rule by query in the flat tab view in Siebel Tools in two ways. Query on the Dock Object Visibility Rule object against the SQL Statement and DBX SQL Statement fields. Or query on the Dock Object Table object against the Filter SQL Statement. If you are in doubt about the nature of a particular column and how it may impact Siebel Remote you should contact Siebel Technical Support.

  For more information about Siebel Remote, see *Siebel Remote and Replication Manager Administration Guide*.

- **Foreign key columns.** If you need to add a new foreign key to a table, when possible reuse a column that exhibits the correct foreign key relationships. This is determined by the Foreign Key Table property of the column object. If the column is already used by a field on the business component, then that field should be reused rather than creating a new field. The original purpose of the unused foreign key field or column should match your intended use.

  You should not use a foreign key column that does not exhibit the correct foreign key relationships, nor should you use an out-of-the-box column that is not a foreign key, to store a custom foreign key. Doing so can impact Siebel Remote and EIM. If no suitable field or column exists, then use Database Extensibility to create a new column and invoke the Siebel Dock Object and EIM Table Mapping Wizards if desired.

  Creating custom foreign keys can cause problems when generating EIM mappings and when routing data to remote users.

- **Primary Id fields.** In certain cases, you can configure primary Id fields for MVLs to improve performance. For a custom MVL, you should attempt to reuse an unused primary Id field or column before deciding to create a new custom extension column.

  The out-of-the-box field's unused column should have the Foreign Key Table property pointing to the table used by the MVL's business component. Also, the Primary Child Col property should be set to TRUE and the Primary Child Table, Primary Child Join Column, and Primary Join Column Name properties should be set with an appropriate value. For many-to-many (M:M) relationships, the Primary Inter Table Name should point to the intersection table. You cannot set these values for out-of-the-box base table columns, so you are simply making sure that the unused field or column is an appropriate primary Id field.

  If an appropriate unused primary field or column is found, you need to verify that it is not already being used by another MVL. Sharing primaries for MVLs that return different result sets is not recommended as it can lead to corruption of the primary. For example, you may have two MVLs to business components against the same table with different search specifications. Sharing a primary Id field for the two MVLs will cause the primary Id field to be populated with the value of "No Match Row Id," since the value of the primary Id field may point to a record not returned by one of the MVLs.

  If no suitable unused primary Id field or column is found, then you should extend the base table and create a custom field. Note that the EIM Table Mapping Wizard does not automatically create EIM Explicit Primary Mapping objects for custom primary Id fields.

  For more information about Primary Id Fields, see "Configuring the Primary ID Field" on page 219.

- **Use of 1:1 ATTRIB columns.** Siebel Systems provides a number of 1:1 tables (for example S_CONTACT_X, S_ORG_EXT_X) which contain generic ATTRIB columns that can be used when no other suitable field or column exists. In certain cases, when a new attribute is required, you should consider using these ATTRIB columns instead. However, there are cases when you should extend a base table with a custom extension column instead:
  - For foreign keys and primary Id fields, the base table should always be extended to aid performance.
  - Columns that are frequently queried or are always present in the result set. For example, fields that are in the initial list views of a screen or fields that have their force active or link specification properties set to TRUE.

  NOTE: When reusing an existing ATTRIB column, make sure that it is not used by another field. If it is, choose another unused ATTRIB column.

  For more information about 1:1 extension tables, see "Using Static 1:1 Extension Tables" on page 112.
- **User key columns or fields.** Fields or columns that are part of the user key of a table should not be used for the purposes other than which they were intended. Doing so could result in performance degradation.

  For more information about performance, see *Performance Tuning Guide*.

  In addition, if the user key column is a foreign key to a table, and you are misusing it by populating a non-foreign key value or a foreign key to a different table, then this may cause issues with EIM. EIM relies on the user key to identify a unique record, and inappropriately populating a user key column based on a foreign key will prevent its effective operation.
- **Bounded LOV Columns.** When columns have the LOV Bounded property set to TRUE, EIM populates the column only if the corresponding LOV type and value exist in the S_LST_OF_VAL table. To reuse a field or column based on an LOV bounded column and to populate it using EIM, make sure that you use a bounded picklist that uses the same LOV type as the column object.
- **Inactivated fields.** You should not reactivate fields that are currently inactivated in standard Siebel configuration. These fields may be obsolete and be deleted in future releases, or they may be part of future functionality that has not been completed by Siebel Engineering.
- **Obsolete columns.** When deciding to reuse an unused column, you should check the Comments property of the column object to determine if the column is obsolete or not. Using such columns should be avoided as they may be deleted in a future release.
- **Fields on specialized business components.** Fields on specialized business components should only be used for the purposes for which they were intended. Specialized behavior may impact on unintended uses of such fields.

Related topics

"About Columns"

"Guidelines for Configuring Business Components"

"About Fields"

About Reusing Business Components

In general, business components in the out-of-the-box repository should be reused as much as possible. However, only those business components whose intended uses have a valid functional and technical fit with your requirements should be reused. For example, you would use the Contact business components to store individual details and the Account business components to store details of the business relationship with a customer. You should not use the Service Request business components to store non-Service Request related entities like financial transactions or order history. In other words, the out-of-the-box use of the business component should match that of your intended use.

When reusing a Siebel business component, you should configure it to be as flexible and reusable as possible. For example, in one release you may implement customer complaints using the Service Request business component and in another release, change of addresses. In both cases, the Service Request business component should be used, rather than cloning the business component in subsequent releases for other service transactions. For example, you could use the "SR Type" field to distinguish between the two service transactions. Your business rules should also be as generic as possible to facilitate the use of a single business component.

Inappropriately reusing a business component may also make it difficult to reuse the same business component for its intended purpose in the future. Using the example given above, if the Service Request business component is used to store financial transactions in one release, it may prevent you from reusing the same business component to store actual service requests in future releases.

When no suitable out-of-the-box business component exists, it may be necessary to create a new one. It may also be necessary to use Database Extensibility to create a new table. For more information see, "Creating Business Components" on page 173 and "About Database Extension Options" on page 109.

There are a number of factors that may influence whether you need to reuse or create a new business component.

- **Specialized classes.** If you use a business component based on a specialized class for a purpose other than the purpose for which it was intended, the class code may result in indeterminate issues that are difficult to debug. For example, you may choose to use the eEvents Parent Event business component to store data unrelated to events management. However, specialized code will cause the automatic creation of sub-events, which is probably undesirable for your custom requirements. Instead of misusing a specialized business component, consider creating a new business component.
- **Party-based business component.** Business components based on the S_PARTY table have significant influence on areas such as access control and remote visibility. They should never be used for purposes other than which they were intended.
- **Licensing.** Unused business components that are not licensed for your configuration cannot be reused.
- **Repository business components.** Repository business components (those that start with "Repository") should never be used by the developer for custom purposes.

Related Topics

"About Business Components" on page 167

"Guidelines for Configuring Business Components" on page 171

About Reusing Tables

If you have decided to create a custom business component because no out-of-the-box business component is suitable, you must also decide whether to reuse an existing table or create a new one.

Again, it is important to only reuse an out-of-the-box table that has a suitable technical and functional fit. There are several reasons for this:

- The table's indexing is tuned for its originally intended use. Using it for alternative purposes may result in performance difficulties.
- The table's user key and unique indexes may not suit your requirements. For out-of-the-box tables, these are not changeable by the developer. Inappropriately populating user key columns may compromise the uniqueness of the record, performance, as well as Siebel EIM.
- The dock object visibility rules may not suit your intended use. Misusing the table may compromise Siebel Remote.

If a table is reused inappropriately, it may make it difficult to reuse that table in the future for its original purpose. For example, you may choose to use the S_CALL_LST table to store data unrelated to call lists. When you decide to implement out-of-the-box list management functionality, unrelated data will be displayed in the list management views. The addition of search specifications to remove this data may compromise performance that may or may not be fixable with the addition of indexes.

The following are some factors that may influence your decision to reuse versus create a new table definition.

- **Overloading of tables.** Occurs when you reuse the same table multiple times on different business components and each business component is "typed" with a search specification. For example, you may choose to use the S_EVT_ACT table to store regular activities, audit logs, error logs, messages, EAI logs, and so forth.

  When you decide to overload a table, it is often necessary to add a search specification against a "Type" field to prevent data from one business component displaying in another.

  Overloading of tables may cause several issues:

  - The search specification used to "type" the table into various business components may cause performance issues. Often, the table is not designed to be overloaded. For example, on the S_EVT_ACT table, the TODO_CD column, which is often used for "typing" the table, is not denormalized on to the S_ACT_EMP activity/employee intersection table and queries using SalesRep visibility against a business component based on the S_EVT_ACT table may result in compromised performance.
  - There is no guarantee that the addition of indexes against these "type" columns will resolve any performance issues because adding one may compromise performance elsewhere. Again, the fact that the "type" columns are often not denormalized onto position, employee, or organization intersection tables will have an impact on queries in certain views.

- Overloading of tables increases the table size.

NOTE: Some tables in the Siebel repository have been built to be overloaded. For example, in Siebel Industry Applications, the S_ASSET table uses the TYPE_CD column to "type" various business components. This column is denormalized and indexed onto the S_ASSET_POSTN and S_ASSET_BU intersection tables to aid performance in SalesRep and Organization visibility views. Also, XM tables such as S_ORG_EXT_XM are built for overloading.

- **1:1 Tables as base tables of business components**. Tables of type "Extension" or "Extension (Siebel)" should never be used as the base table of a business component. Siebel EIM and Remote assume that the PAR_ROW_ID and ROW_ID columns on these tables are equivalent and that the PAR_ROW_ID column points to a valid parent table record.

  For more information about 1:1 extension tables, see "About Extension Columns" on page 105.

- **New 1:1 tables.** There are rare instances when you would need to create a new 1:1 table. In most cases, you would simply extend the base table or reuse an ATTRIB column on a 1:1 _X table. However, you may need to create a new 1:1 table to add LONG columns, since they cannot be added to base tables and only one LONG column is supported on a given table.

  For more information about LONG columns, see "Creating Columns of Type LONG" on page 115.

- **New 1:M tables.** In most cases you would simply reuse an existing XM table to support a 1:M relationship off a base table. The following guidelines should be followed when using an XM table:

  - There are very few cases where you would need to create a new XM table. XM tables are already tuned to support large data volumes and of multiple data types.
  - In some instances, a base table does not have any XM tables. Instead of reusing another unsuitable out-of-the-box table merely because it has a foreign key to the base table, it may be necessary to use database extensibility to create the new table. For example, if you require a one-to-many (1:M) business component from S_EVT_ACT, you should create a 1:M table rather than reusing a table that may be inappropriate, such as S_ACT_TIMESTAMP, provided that the business component is not storing timestamp information.

  For more information about 1:M extension tables, see "About Extension Tables" on page 92.

- **Intersection tables.** You should reuse an intersection table only where it is a true intersection between the two tables. The table should also be of type "Data (Intersection)".

  You should not use a non-intersection table as an intersection table just because it contains foreign keys to the desired tables. This results in an overloading of the table. For more information about overloading tables, see "About Object Reuse" on page 51.

  Also, 1:M XM tables should not be used as intersection tables. Performance of an XM table is not tuned with this in mind, and using it as an intersection table may cause performance degradation. In addition, a custom foreign key will most likely need to be created to support one side of the relationship, resulting in issues with Siebel Remote and EIM.

  Where no suitable intersection table exists between two tables and one is required, it would be necessary to use DB Extensibility to create it.

  For more information about intersection tables, see "About Extension Tables" on page 92.

- **Licensing.** Unused tables that are not licensed for your configuration cannot be reused.

- **Custom party types.** The S_PARTY table should not be reused to support custom party types. The S_PARTY.PARTY_TYPE_CD column only supports one of the following: AccessGroup, Household, Organization, Person, Position, UserList. Use of custom party types will compromise access control and remote visibility.

  For more information about S_PARTY, see "About the S_Party Table" on page 108.

- **Repository tables.** Repository tables should not be misused by the developer. Repository tables are those that are used by business components prefixed with "Repository."

About Reusing Views

When deciding whether to reuse views or create new views, consider the following:

- When the requirements for a new view closely align with an existing view, but call for simple changes such as changing the view title, or moderate layout changes, such as displaying a different applet or adding toggles, then modify the existing view.
- For views that consolidate two existing views, it is recommended that you configure one of the existing views by modifying the existing view object, and then removing visibility to the redundant view using the Responsibility Administration screen.
- When the requirements for a view do not align with any existing views, create a new view. Typically these are views that expose new functionality configured for your implementation. For example, you may need a view to expose new business objects or business components. In these cases, creating a new view rather than modifying an existing one, will be easier to maintain and upgrade (both manually and automatically).

When copying objects, use the Upgrade Ancestor property. For more information, see "About Upgrade Inheritance" on page 50.

Related Topics

"About Views" on page 281

"Guidelines for Configuring Views" on page 284

About Reusing Applets

When deciding whether to reuse applets or create new applets, consider the following:

- Configure existing applets when the requirements closely align with an existing applet but require minor configuration, such as a title change, inactivating or adding controls or list columns, or changing display labels.
- Create new applets (by copying a similar applet) when the applet represents an existing relationship (for example, opportunity contacts), but requires modification of the applet layout, such as resequencing, inactivating, and adding controls and list columns. The resulting configuration will be much cleaner and easier to maintain and upgrade (both manually and automatically).

- Create a new applet (by copying an similar applet) when the applet requires different drilldowns in different views.
- Create new applets when there is no equivalent existing applet. These tend to be applets that expose a new business component.

When copying objects, use the Upgrade Ancestor property. For more information, see "About Upgrade Inheritance" on page 50.

Related Topics

"About Applets" on page 231

"Guidelines for Configuring Applets" on page 236

About Scripting and Object Interfaces

Siebel applications are primarily enhanced by creating and modifying object definitions. However you can also create scripts to enhance Siebel applications using Siebel Visual Basic (VB), Siebel eScript, and Browser Script.

NOTE: Scripts are associated with the Siebel Event Model. Each script is associated with a specific object and event.

Related Topics

"Server-Side Scripting" on page 62

"Browser-Side Scripting" on page 63

"Generating Browser Scripts" on page 65

Server-Side Scripting

You can use Siebel VB and Siebel eScript to write event procedures, known as scripts, which are attached to object definitions of specific object types. (These object types include Application, Business Component, Business Service, and Web Applet.) For example, these custom event procedures can be used to attach additional validation logic to a business component.

The Script Editor, Debugger, and Compiler are used to create and test Siebel VB, eScript scripts, or browser scripts to extend and further configure Siebel applications. This capability is integrated with the Siebel Web Applet Editor, so you can attach scripts to user interface element controls like fields and ActiveX controls.

- **Siebel VB.** Siebel VB is a language provided in Siebel Tools that is similar to Microsoft Visual Basic. You can write event procedures to attach to certain object types. For example, these custom event procedures can be used to attach additional validation logic to a business component.

  NOTE: Siebel VB is available on Windows platforms only.

- **Siebel eScript**. Siebel eScript is a JavaScript-compatible language provided in Siebel Tools that is used to write event procedures. Siebel eScript supports scripting in Windows as well as non-Windows environments such as UNIX.
- **Simultaneous Use of eScript and VB Script.** You can write scripts that respond to various client-side events using Siebel VB and Siebel eScript simultaneously in the same environment. VB and eScript can be used concurrently (but not within the same object). It is recommended that you use eScript alone because it works on UNIX as well as Windows servers. When you initially add script to an object, you are prompted to choose the scripting type.

  Siebel VB and Siebel eScript can be used to program the following kinds of enhancements:

  - **Data validation routines.** These routines enforce specific business rules before or after performing record manipulation operations. Validation routines are performed before the user performs an update or an insert. The intent is to make sure that illogical or incomplete data is not entered into the database.
  - **Data manipulation and computational routines.** These routines can be used to modify or analyze data.
  - **Data transport routines.** These routines import and export small volumes of data between Siebel applications and other applications.
  - **Application launching routines.** These routines launch external applications on the server side in response to Siebel events, and pass start-up parameters. Valid for browser scripting only.

The Siebel VB and Siebel eScript development environments provide you with a programming platform to:

- Integrate Siebel applications with third-party cooperative applications
- Extend the base functionality of the Siebel application screens and business components

For more information about Siebel eScript and Siebel VB, see *Siebel eScript Language Reference* and *Siebel VB Language Reference*.

NOTE: Scripts already written for prior releases of Siebel eBusiness Applications can be redeployed in the Siebel Web Client. However, some modifications may be required. For more information, see the *Upgrade Guide* for the operating system you are using.

Browser-Side Scripting

Browser script allows you to extend browser behavior using JavaScript, an interpreted language that runs in many Web browsers. Browser scripts respond to events on browser-side Java objects. These browser objects work in tandem with their corresponding objects running in the object manager.

Browser scripts are written using Siebel Tools and can be associated with the following Siebel object types: applets, business components, business services, and applications. See Figure 7 on page 64.

Like their server-side counterparts, browser script object types allow you to write event procedures. However, the set of events that can be scripted with browser object types are different from their server-side counterparts. Browser-supported events can be scripted for the following cases:

- For Siebel customer applications, a wide array of browser-supported events can be scripted. However, the OnClick event is not supported for HTML controls.

  See *Siebel eScript Language Reference* for detailed information.

- For Siebel employee applications, the OnBlur and OnFocus events are the only control events that can be scripted.

NOTE: In standard interactivity, the following Siebel objects are not available for browser scripting: applet, application, business component, and business service. You cannot write script to handle pre- and post-events. However, you can write scripts to handle control-level events such as Onclick, Onblur, and Text controls.

For scripting against browser-side events, you are provided with Browser Script children object types of the Application, Applet, Business Component, and Business Service object types. These object types are illustrated below, along with their server-side (Script and Web Script—for scripting in Siebel VB, JavaScript, or Java) counterparts. Figure 7 shows the various scripting objects.

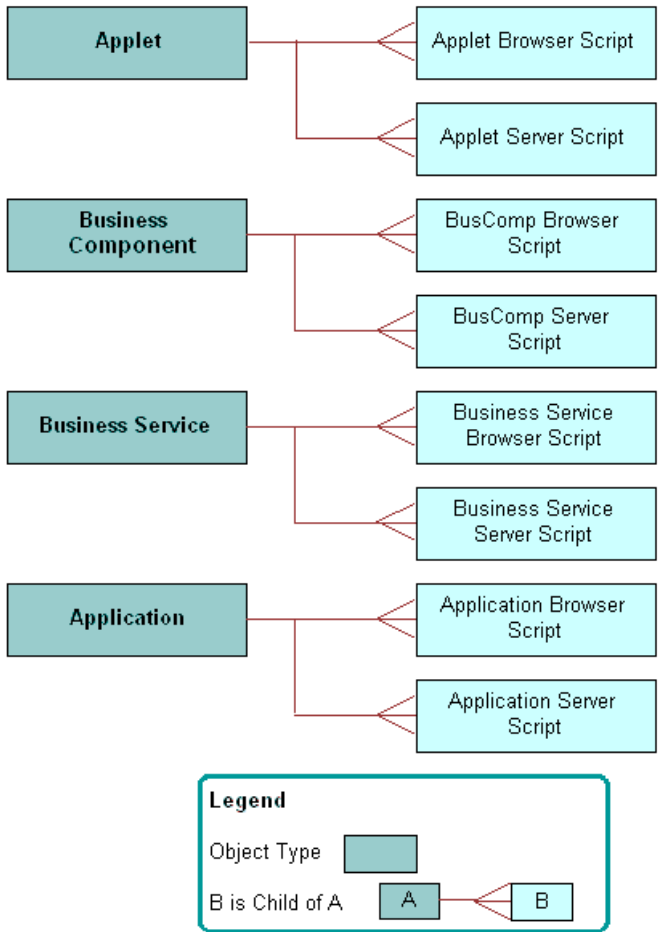


Figure 7. Scripting Object Types

The following example, shown in Figure 8, illustrates how you can configure client-side form field validation using Browser Script. It shows browser script for the Account BrowserBusComp PreSetFieldValue event handler.

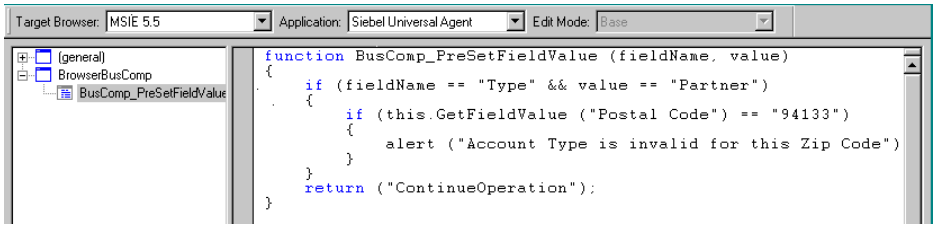


Figure 8. Example Browser Script

NOTE: For employee applications that use the High Interactivity Framework, business component browser script is appropriate when only active objects (that is, Siebel objects exposed on the UI) are used in the script.

For standard interactivity applications, Browser Script must be written on the control's onChange browser event and must use the native methods of the browser Document Object Model (DOM). Each control associated to an applet can be scripted for standard browser events, such as onChange, onMouseOver, onFocus, onMouseOut, and onBlur.

Generating Browser Scripts

Browser scripts are JavaScript files (.js) that are generated in two ways.

- Browser scripts are automatically generated when you compile objects to a repository file.

  When you compile objects to a repository file, browser scripts are generated for compiled objects only. They are placed in the directory specified in the Development Tools Options dialog (View > Options > Scripting).

  If you do not specify a directory, browser scripts are stored in the following default directory:

  tools_root\public*language_code**srf_timestamp*\bscripts\all

- Browser scripts can be generated using the genbscript.exe utility from the command line interface.

  When you run genbscript.exe, all browser scripts in the repository are generated. They are placed in a directory that you specify using the destination directory parameter (dest_dir). The genbscript.exe utility is located in:

  siebsrv_root/bin or *client_root*/bin

  The syntax for running genbscript is:

  genbscript *config_file dest_dir* [*language*]

  For example:

```
genbscript c:\sea15022\client\bin\enu\uagent.cfg c:\sea15022\client\public\enu
enu
```

NOTE: The language parameter is optional for ENU, but must be specified for other languages.

Browser scripts must be deployed in the following directories on the Siebel Server and the Siebel Mobile Web Client:

- On the Siebel Server, browser scripts must be located in:

  siebsrvr_root\webmaster

- On the Siebel Mobile Web Client, browser scripts must be located in:

  client_root\public*language_code*

When browser scripts are generated, a directory path is created and named according to the version of the repository file (.srf). It is appended to the path specified as the destination directory. For example, after compiling browser scripts to the correct location on the Siebel Server, the complete path to the browser script files would be:

siebsrvr_root\webmaster*srfTimestamp*\bscripts\all\

If you are migrating scripts from one location to another, you must be sure to copy the directories (\srfTimestamp\bscripts\all\) to the correct location.

NOTE: If you are migrating browser scripts to a Siebel Server running on Solaris or AIX, after compiling on a Windows machine, you must FTP the directories to the correct location on the UNIX machine.

After you compile browser scripts you must do one of the following to load the scripts into the Siebel Web server extension, otherwise you may receive an *Object Not Found* error message:

- Stop and restart the Web server.
- Log into the application (for example, http://*user_name*.siebel.com/callcenter) and then type the following URL into the Address field of the browser:

  http://*user_name*.siebel.com/callcenter/
  start.swe?SWECmd=UpdateWebImages&SWEPassword=*passwd*

  Where *passwd* is the WebUpdatePassword from the eapps.cfg file.

About Localization

Translatable text strings and language-specific attributes of Siebel objects are maintained in the same repository. Siebel Tools editors allow you to edit the locale-specific attributes of objects, such as Applets, Views, and Controls in multiple languages.

For detailed information about localization, see *Using Siebel Tools* and *Global Deployment Guide*.

About Locale Object Types

For all Siebel object types that contain localizable data, such as symbolic strings, there are child locale objects used to define the locale-specific data for the parent object.

Locale object types store their data in a set of repository tables used specifically for storing locale-specific data. These tables follow a naming convention that includes the name of the base table followed by the suffix *_INTL*.

For more information about working with locale-specific data, see *Using Siebel Tools*.

About the Siebel Tools Language Mode

To determine what localizable data to use with translatable attributes, Siebel Tools runs in a particular language mode. Siebel Tools itself runs in an English-American user interface, but you can edit localizable data in the language that you wish. English-American is the default edit language. Siebel Tools has a language mode that can be set from the Development Tools Options dialog box.

NOTE: If additional languages are added to the system, the language code must be in all capital letters.

For more information about the Tools Language mode, see *Using Siebel Tools*.

About Checking Out and Checking In Locale-Specific Data

The server keeps track of the language in which the project was checked out. This information is shown in the Server Language column in the Check Out dialog box. This feature allows your team to work with language-specific data in languages other than the ones in which the project has been checked out.

You also can get locale specific data for a particular project. You can do this when you decide to change your current working language. For example, if you only have language-specific data for English-American (your current working language in Siebel Tools) and you decide to switch to French, unless you get the locale-specific data for French you will not be able to view any localizable data in Siebel Tools.

For more information about check in and check out, see *Using Siebel Tools*.

Locale Management Utility

The Locale Management Utility (LMU) allows you to export and import text strings and locale-specific information to an external file. This is typically used to export strings to send out for translation and then to import the translated strings back into the repository. It facilitates a concurrent application configuration and localization process.

The primary users for this option are customers deploying in multiple languages. This utility can be invoked from the Repository menu by choosing Tools > Utilities > Locale Management option.

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

Other Ways to Customize Application Behavior

Configuring object definitions using Siebel Tools is not the only way to customize Siebel applications. There are several other mechanisms that allow you to customize business logic, such as Siebel Personalization, Siebel SmartScript, Siebel Business Process Designer, Siebel Assignment Manager, State Model, and Siebel ePricer. These areas of functionality are controlled through run-time client administration views rather than Siebel Tools and are used by both developers and administrators.

Personalizing Your Web Application

Personalization allows you to target content within applets that are directed to particular users based on their preferences or profile. For example, you can include a salutation applet in your application that greets the user by name, indicates how long it has been since the user last visited the site, and presents the user with information about specific products or services you believe the user may be interested in.

Some key points about personalization:

- Personalization is available on any applet in your Web application.
- Personalization uses rules and rule sets to specify which records a user should see in a given applet, based on the user's profile. Rules evaluate profile attributes to determine which records and applications to display. A rule set is a group of rules. If desired, you can create multiple rule sets such that if none of the criteria in one set of rules is met, the next rule set is evaluated.
- The user profile is based on any attribute that belongs to a Contact and the contact's account (if the user is a contact) or any attribute that belongs to an Employee and the employee's division (if the user is an employee).
- Personalization uses a new object called the User Profile Attributes to hold and retrieve elements of a user profile. These attributes can be used for display in the user interface of the Web application, and in Rules that determine the content users see.
- You can track events that occur in the context of the Web application. Specifically Siebel Personalization can track application, business component, and applet events. When an event occurs, it triggers a Personalization Action which modifies a user's profile attributes.
- Actions can be called by a rule or on an event. An action is used to set either a predefined profile attribute or a profile attribute created dynamically during run time. Profile attributes created dynamically during run time only exist for the duration of the user session. Profile attributes that are configured in Siebel Tools and those that are created during run time can be used to store state information in much the same way that variables stored in cookies or persistent frames might be. Wherever possible, profile attributes should be used in placed of cookies.
- Rules or actions can invoke business component methods or business services methods. Typically, these methods are used to return values that can be used either as criteria for a rule, or for setting a profile attribute.

For more information about personalization, see *Personalization Administration Guide*.

Managing Web Content with Siebel eBriefings

Siebel Interactive technology allows customers to incorporate HTML documents stored either on the same or on a different Web site. By configuring business components based on the CSSBCExternalUrl and CSSBCVExternalUrl classes, and setting the appropriate field, field user property, and configuration file parameters, configurators can implement functionality for retrieving and displaying internal or external HTML content. Through this type of configuration, you can programmatically forward and execute search specifications against desired Web servers. This functionality is also ideal for managing large stores of internal HTML-based content which may have informational value for users (for example, FAQs and so on).

For more information about eBriefings, see *Siebel eBriefings Guide*.

Dynamic Data Capture with Siebel eSmartScript

Siebel eSmartScript allows you to deploy an interactive guide (question and answer format) in a Web page that helps users find information. The interactive guide continually asks users to answer questions to refine their search. Based on their answers, the guide continues down branching paths to find the correct answers. Siebel eSmartScript is completely integrated with Siebel SmartScript to allow you to define scripts using a single administrative user interface, and then deploy those scripts to either call center agents or to end users through the Web.

Siebel eSmartScript is configured through the same administrative screens used by SmartScript.

Existing SmartScripts can be deployed with little or no additional configuration. Application configurators and administrators need only expose the eSmartScript view, and the rest of the views, applets, and so on will be generated dynamically.

Siebel eSmartScripts can dramatically simplify Web configuration by making applications more data-driven. Additionally, Siebel eSmartScripts are relatively easy to configure, deploy and administer.

For more information about Siebel eSmartScript, see *Siebel SmartScript Guide*.

Siebel Assignment Manager

Siebel Assignment Manager is used to assign the most qualified people to specific tasks. This is accomplished by matching candidates to predefined and user-configurable assignment objects. To assign the most qualified candidate to each object, Siebel Assignment Manager applies assignment rules that you define.

You can customize the way Assignment Manager makes assignments by defining how attributes will be matched or by creating and configuring your own components. Assignment Manager can be run in different modes to process assignments interactively in real time, dynamically when database changes are made by connected or mobile users, or periodically assigning objects in batches. For more information on Siebel Assignment Manager, see *Siebel Assignment Manager Administration Guide*.

Siebel Business Process Designer

Siebel Business Process Designer uses as its basic model the processes that organizations use in their sales, marketing, and service departments that determine business workflow. You can use Siebel Business Process Designer to assure consistency and adherence to agreements through the automatic enforcement of business policies and procedures. Siebel Business Process Designer is a customizable business application providing the capability to manage and enforce business processes such as response time objectives, specifying review policies, and monitoring service requests or opportunities over time. For more information, see *Siebel Business Process Designer Administration Guide*.

State Model

The state model provides a data-driven method for extending workflow control based on the status of an object such as a service request or a product defect.

A *state model* is the blueprint of acceptable states and state transitions that the state machine enforces. The state machine then makes sure that these objects go through the desired process defined in the state model.

A *state machine* is an engine that enforces the transitions between states for an object during its lifetime. A state represents the status of an object, such as Open, Closed, or Pending. The state represents where the object is in its lifetime. The state can also control whether or not the data of that object can be modified. For example, a service request that is in a Closed state may be considered frozen, such that its attributes cannot be modified.

A *state transition* defines the allowable migration of an object from one state to the next. For instance, a service request that has been closed but must be re-opened may go from the Closed state to an Open state, and may go from Open to Pending, but may not transition directly from Closed to Pending. The allowable migration of a service request from Closed to Open, or Open to Pending, represents defined state transitions.

State Model is administered through the Siebel Business Process Designer on the Siebel client. For more information on accessing and using the State Model views, see *Siebel Business Process Designer Administration Guide*.

Siebel ePricer

Siebel ePricer provides a solution for creating, assessing, administering, and deploying flexible pricing strategies. Siebel ePricer consists of the following:

- A set of administration views that allow you to define pricing adjustments and the conditions under which they should be applied.
- An engine that evaluates the condition statements and determines which pricing adjustments should be applied.
- A testing area that allows assessment of the pricing adjustments.

- Integration with end-user interfaces, such as Quotes, Orders, Siebel eSales, Siebel PRM, and Siebel eConfigurator.

Siebel ePricer is composed of the following components:

- **Price lists.** Contain base prices.
- **Pricing models.** Management tool to control a set of related pricing factors.
- **Pricing factors.** Statements that define conditions and pricing adjustments.
- **Scripting.** Lets you use business services with a pricing factor to extend the pricing calculation and to access external data.
- **Pricing validation.** Lets you test pricing factors and the pricing model before releasing for use by end users.
- **Reports.** Lets you print reports of pricing factors.
- **Pricer Engine.** Evaluates conditional statements and applies pricing adjustments.

For more information on accessing and using the State Model views, see *Pricing Administration Guide*.

Setting Up Developers as Mobile Users

After installing Siebel Tools, the Siebel server, and other necessary software in the development environment, developers need to be set up as mobile users. As mobile users, developers can store copies of the Siebel database (including the Siebel repository) on their local machines. They can check out objects from the server repository, configure and test on their local machines, and then check objects back into the server.

For information about installing software, see *Siebel Installation Guide* for the operating system you are using.

For more information about the check in and check out processes, see *Using Siebel Tools*.

To set up developers as mobile users

1. Install Siebel Tools on all developers' machines, in a directory separate from the Siebel Client.

   For example, if you have installed your Siebel Client in C:\siebel\clnt, install Siebel Tools in C:\siebel\tools. This makes sure that the development and run-time environments are distinct and makes sure that if you use Siebel Remote in both environments, the two installations do not conflict.

   For more information about installing Siebel Tools, see *Siebel Installation Guide* for the operating system you are using.

2. Verify that each developer has a valid user name and password for the Siebel development database server.

   In most cases, their employee logins and passwords will also be their database server user names and passwords.

3 Using a Siebel application client connected to the development server database, create an Employee record and a Mobile User record for each developer.

Use the developer's first and last names for the employee first and last names. Use a standard naming convention, such as first initial and last name, for the logon name. This makes it easy to identify who has locked a project.

NOTE: Password encryption interferes with project check-in and checkout. You must disable password encryption in the client or configuration file when running Siebel Tools if you will be checking projects in and out.

For detailed instructions on setting up mobile users, see *Siebel Remote and Replication Manager Administration Guide*.

4 Grant each developer a position and a responsibility.

Grant each developer the Developer and Siebel administrator responsibilities. You may also create a responsibility with access to all views except the System, Service, and Marketing Administration views to prevent unintended changes to important system preferences. You can use a common position for all developers, but, for testing purposes, you should also set up an organization structure that models the business.

NOTE: If you do not grant the user the Developer responsibility, drilldowns will not be activated in the Tools client.

For more information on setting up employees, see *Applications Administration Guide*.

For more information on setting responsibilities, see *Security Guide for Siebel eBusiness Applications*.

5 On the Siebel Server, generate a database template.

Set the value of the SQL Anywhere Database parameter to sse_utf8_internal.dbf. The database template sse_utf8_internal.dbf is used for developers only; the default database template is used for Mobile users is sse_utf8.dbf.

NOTE: If sse_utf8_internal.dbf does not exist in the SIEBEL_ROOT\dbtmpl directory, copy sse_utf8.dbf and rename the copy to see_utf8_internal.dbf.

For detailed instructions on how to generate database templates and set component request parameters, see *Siebel Remote and Replication Manager Administration Guide*.

6 On the Siebel Server, extract each developer's local database using the Database Extract component.

Database Extract creates a template for the developer's local database that is populated only with business data, not repository data. All enterprise-visible data is extracted into this template, together with any limited-visibility data (contacts, accounts, opportunities, and so on) to which this user has access.

For detailed instructions on how to run Database Extract server task, see *Siebel Remote and Replication Manager Administration Guide*.

7 Initialize the Developer's Mobile Client Database by double-clicking the Siebel Tools icon and connecting to the local database.

A message appears saying that the local database is not found.

8 To start the initialization process click Yes.

9 In the Siebel Remote Parameters dialog box, enter the Siebel developer logon created in Step 3 on page 72 and an appropriate password.

The initialization program creates the sse_data.dbf local database in the \local directory of your Siebel Tools installation, for example c:\siebel\tools.

10 Do an initial get of all projects on each local database.

For more information about performing a Get, see *Using Siebel Tools*.

3 Working with the Entity Relationship Designer

Topics in This Chapter

About the Entity Relationship Designer

The Entity Relationship Designer is a visual design tool that you use to create entity relationship diagrams (ERDs) to represent your business. You then map the entities and relationships depicted in the diagram to objects in the Siebel repository, such as business components, links, and joins.

The Entity Relationship Designer provides a drag-and-drop environment for creating ERDs. It allows you to show cardinality between entities using the diagraming convention commonly called *crows feet*. It provides various edit and layout options, such as aligning shapes, moving shapes, and modifying text. When binding entities and relations to Siebel objects, the Entity Relationship Designer filters the list of objects that you choose from to include only those objects that support the context represented in the ERD. When no business components are suitable for binding, you can launch a wizard from within the Entity Relationship Designer to assist you in creating a new business component.

Using the ERD to diagram business entities and map them to Siebel objects has the following benefits:

- Requires less work to define data model requirements.
- Initial mapping of data requirements to the Siebel data model is more efficient and effective because lists of components are constrained to only those that fit the context described in the ERD.
- Improves your ability to trace configuration changes back to data model requirements.
- Creates a permanent record of ER design in the Siebel repository.

Navigating to Entity Relationship Diagrams

You navigate to ERDs using the Object Explorer and Object List Editor in Siebel Tools.

To navigate to an entity relationship diagram

1. In the Object Explorer, select the Entity Relationship Diagram object type.
2. Select a record in the Entity Relationship Diagram list.
3. Right-click and then choose Edit Entity Relationship Diagram.

   The Entity Relationship Diagram Canvas appears.

Process of Creating and Binding Entity Relationship Diagrams

Two people usually perform the process of creating and binding ERDs. A business analyst creates ERDs that represent the customer's business and a technical architect or developer binds the entities and relationships in the diagram to Siebel objects. The business analyst is knowledgeable about the customer's business and the system architect or developer is knowledgeable about the Siebel data model.

To create entity relationship diagrams and bind them to Siebel objects, perform the following tasks:

"Creating Entity Relationship Diagrams" on page 77

Creating Entity Relationship Diagrams

You can create entity relationship diagrams that represent your business. ERDs include business entities, entity attributes, and the type of relationship (1:1, 1:M, M:M) that exists between entities. Examples of common business entities include accounts, contacts, and addresses.

This task is a step in "Process of Creating and Binding Entity Relationship Diagrams" on page 76.

To create entity relationship diagrams

1. In the Object Explorer, select the Entity Relationship Diagram object type.
2. In the Entity Relationship Diagrams list, right-click, and then choose New Record.
3. Enter a Name and associate a Project to the new record.

   A project must be locked to be able to enter it in the Project field.

   For more information about projects, see *Using Siebel Tools*.
4. Right-click and choose Edit Entity Relationship Diagram.

   The Entity Relationship canvas and shape palette appears.
5. Drag and drop Entity shapes from the palette to the canvas.
6. Drag and drop the desired Relationships (1:1, 1:M, M:M) from the pallet to the canvas, positioning the ends of the relationship lines on entity shape connector points.

   NOTE: When a relationship line is selected in the ERD canvas, the Entity Physical Relations list appears. This list is populated after binding relationships to Links and Joins. See "Binding Relationships to Links or Joins" on page 80.

Defining Entity Attributes

After you have created an ERD that includes entities, you can define attributes for each entity.

This task is a step in "Process of Creating and Binding Entity Relationship Diagrams" on page 76.

To define entity attributes

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, select an entity.

3 In the Entity Attributes list, right-click and then choose New Record.

4 Enter a Name and a Data Type for the entity attribute.

Binding Entities to Business Components

After you have created an ERD that includes entities, you can bind those entities to Siebel business components. When binding business components, the Entity Relationship Designer filters the list of business components to choose from so it includes only those with characteristics that fit the context described in the ERD.

To understand which business components are available to bind to entities, consider the example shown in Figure 9.

Figure 9. Example ERD

Suppose that you want to bind Entity C to a Siebel business component. The Entity Relationship Designer filters the business components available to bind based on the context described in the ERD. Table 9 lists several possible situations and the business components that would be available to bind to Entity C, shown in Figure 9.

Table 9. Business Components Available for Binding

Entity A	Relationship	Entity C	Business Components Available for Binding
Unbound	Any	Unbound	All business components
Bound	1:1	Unbound	Business components with joins to the primary tables of the business component bound to Entity A.
Bound	1:M	Unbound	Business components with a link to the business component bound to Entity A, where the business component bound to Entity A is the parent. Business components with joins to the primary table of the business component bound to Entity A.

Table 9. Business Components Available for Binding

Entity A	Relationship	Entity C	Business Components Available for Binding
Bound	M:1	Unbound	Business components whose primary table is the table joined to the business component bound to Entity A. Business components that have links with the business component bound to Entity A, where the business component bound to Entity A is the child.
Bound	M:M	Unbound	Business component that belong to the intersection of the 1:M and M:1 examples above.

This task is a step in "Process of Creating and Binding Entity Relationship Diagrams" on page 76.

To bind entities to Siebel business components

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select an entity.
3. Right-click and then choose Bind Business Component.
4. In the Bind Business Component dialog box, choose the Siebel business component that you want to bind to the entity on the ERD.

   If no business component meets your needs, you can click New to launch the New Business Component Wizard to create one. For more information, see "Creating Business Components" on page 173.

   After you click OK, the names of the business component and the underlying base table appear in the ERD.

Associating Entity Attributes to Business Component Fields

After binding entities to business components, you can associate entity attributes to business component fields.

This task is a step in "Process of Creating and Binding Entity Relationship Diagrams" on page 76.

To bind entity attributes to Siebel business component fields

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select an entity.

3 In the Entity Attribute list, select the attribute that you want to bind to a business component field.

4 In the Business Component field, click the drop-down arrow, and then select a field.

The entity attribute record is associated with the business component field.

Binding Relationships to Links or Joins

After two entities are bound to Siebel business components, you can bind the relationship between them to a Siebel Link or Join objects. For more information about Links and Joins, see "Configuring Links" on page 205.

The Entity Relationship Designer filters the list Links and Joins available for binding based on the context described in the ERD. To understand which Links and Joins are available to bind to entities, consider the following example (Figure 10):

Figure 10. Example ERD

Suppose that Entities A and C are bound to Siebel business components and that you want to bind the relationship AC to a Siebel Link or Join. The Entity Relationship Designer filters the list of objects available for a particular relationship. Table 10 lists the possible situations and describes the objects that would be available to bind to Relationship AC.

Table 10. Links and Joins Available for Binding

Relationship A-C	Objects Available to Bind
1:M	Joins originating from the business component bound to Entity C to the primary table of the business component bound to Entity A. Links between the business component bound to Entity A and the business component bound to Entity C, where the business component bound to Entity A is the parent and the business component bound to Entity C is the child.

Table 10. Links and Joins Available for Binding

Relationship A-C	Objects Available to Bind
M:1	Joins originating from the business component bound to Entity A to the primary table of the business component bound to Entity C. Links between the business component bound to Entity C and business component bound to Entity A, where C is the parent and A as the child.
M:M	All links between business components bound to Entities A and C.

This task is a step in "Process of Creating and Binding Entity Relationship Diagrams" on page 76.

To bind relationships to links or joins

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select a relationship between entities.
3. Right-click and then choose Bind Entity Relation.

   NOTE: The two entities that the entity relationship line joins must be bound to Siebel business components.

   Existing joins or links between the two business components are displayed in the Bind Relationships dialog box.

4. Select the join or a link that best represents the relationship described in your ERD.

   After the binding is complete, the line turns to a bold line in the ERD canvas.

Viewing the Entities and Relations Lists

After you click the ERD canvas background the Entities list is displayed by default. However, you can toggle between the Entities List and the Relations List using the drop-down list at the top of the list applet.

To view entities list for a given ERD

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, click the canvas background.

   The Entity list appears below the ERD canvas.

To view the relations list for a given ERD

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, click the canvas background.

   The Entity list appears below the ERD canvas.

3. Select Relationships from the drop-down list.

   The Entity Relations list appears.

Modifying Relationship Properties

You can modify the properties of a relationship using the Object List Editor or the Properties Window. For example, you may want to change the text that appears at the end points of a relationship line.

NOTE: You cannot modify cardinality using the Object List Editor or the Properties window. If you want to change the cardinality, delete the relationship in the canvas and drag a new one on to it.

To modify relationship properties using the Entity Relationship list

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select the canvas background.
3. From the drop-down list, select relationship.
4. In the Relationship list, select the relationship you want to modify.
5. Change the record as needed.

   If you change the value for the Name, End Name 1, or End Name 2, the text is updated in the diagram.

To modify the properties of a relationship using the Properties Window

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select the relationship you want to modify.
3. Select View > Windows > Properties.

   The Properties Window appears.

4. Modify the values for the properties as needed.

   If you change the value for the Name, End Name 1, or End Name 2 fields, the text is updated in the diagram.

Modifying Shape Properties

You can modify the appearance of shapes and lines on the ERD canvas. For example, you can highlight an entity by changing the shape's fill colors to something that distinguishes it from the rest.

To modify entity or relationship properties

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select an entity or relationship.
3. Right-click and then choose Shape Properties.

   The Item Properties dialog appears.

4. Select the properties you want to appear and then click OK.

Aligning Shapes

The Align option allows you to align shapes relative to each other on the ERD canvas.

To align shapes in the ERD canvas

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select two or more entities by holding down the shift key while clicking the shapes.
3. Right-click and then choose Layout > Align > and one of the following options:
   - Lefts
   - Centers
   - Rights

Making Shapes the Same Size

You can use Layout option to make one or more shapes the same size.

To make shapes the same size

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, select two or more entities by holding down the shift key while clicking the shapes.

3 Right-click and then choose Layout > Make Same Size > and one of the following options:

- Width
- Height
- Both

Adding Points to Lines

You can change the shape of lines, add right angles for example, by adding points to relationship lines. This avoids overlapping lines when a diagram becomes complex.

To add points to lines

1 Navigate to an entity relationship diagram.

For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, select a relationship line.

3 Right-click and choose Edit > Add Point.

4 With your pointer, grab the point and drag it to a new position on the ERD canvas.

Hiding Line Text

You can hide text associated with a relationship line, including the Relationship Name, End Name 1, and End Name 2.

To hide line text

1 Navigate to an entity relationship diagram.

For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, select an entity relationship.

3 Right-click and choose Edit > Hide Text.

Moving Line Text

You can adjust where the Relationship Name text appears on a line.

NOTE: You cannot adjust where the text for End Name 1 and End Name 2 appears.

To move text

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select a relationship line.
3. Right-click and choose Edit > Move Text Back or Move Text Forward.

Returning Line Text to the Default Location

After moving line text, you can return it to its default location.

To return text to its default location

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select a relationship line.
3. Right-click and choose Edit > Move Text to Default.

Moving Shapes Around the ERD Canvas

You can move shapes around the ERD canvas by dragging and dropping them or by using the options available from the Layout menu.

To move entities on the ERD canvas

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select the shape you want to move and then do one of the following:
   - Drag the shape to another position on the canvas.

- Right-click and then choose Layout > Move and then choose one of the options listed in the following table:

Menu Option	Description
Left by 1	Moves the shape in the selected direction by 1 pixel.
Right by 1	
Up by 1	
Down by 1	
Left by X	Moves the shape in the direction selected by the number of pixels contained in one cell on the ERD canvas. The number of pixels can vary depending on the resolution setting of your monitor.
Right by X	
Up by X	
Down by X	

NOTE: You can also use the shortcut keys displayed in the Layout > Move menu.

Resizing Shapes

You can resize shapes by dragging them in the ERD canvas or using the Resize options from the Layout right-click menu.

To resize shapes

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. In the ERD canvas, select the shape you want to resize and then do one of the following:

   - Click an entity's connection point and then drag it to a new position.

- Right-click and then choose Layout > Resize and choose one of the following options:

Menu Option	Description
Height by 1	Resizes the selected dimension by 1 pixel.
Height by -1	
Width by 1	
Width by -1	
Height by X	Resizes the dimension selected by the number of pixels contained in one cell on the ERD canvas. The number of pixels can vary depending on the resolution setting of your monitor.
Height by -X	
Width by X	
Width by -X	

NOTE: You can also use the shortcut keys displayed in the Layout > Expand menu.

Zooming In and Out

You can zoom in and out of an ERD diagram by a default amount or by entering a percentage.

To Zoom in and out

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2. Do one of the following:
   - Right-click and then choose Zoom In or Zoom Out.
   - Right-click, choose Zoom, and then select a percentage.

Showing Connection Points

Connection points are points on an entity shape where a relationship line connects. You can show connection points or hide them. For example, you can show connection points when creating ERDs and then hide them when printing ERDs.

To show connection points

1. Navigate to an entity relationship diagram.

   For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, right-click and then choose Connection Points.

NOTE: Hide connection points by selecting Connection Points again to remove the check mark.

Showing Grid Lines

Grid lines help you align shapes and lines on and ERD. It is useful to show grid lines when working with an ERD and then hide them when you print it.

To show grid lines

1 Navigate to an entity relationship diagram.

For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, right-click and then choose Show Grid.

NOTE: Hide Grid Lines by selecting Show Grid again to remove the check mark.

Turning Snap to Grid On

Snap to Grid helps you keep shapes and lines aligned as you draw your ERD.

To turn on snap to grid

1 Navigate to an entity relationship diagram.

For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, right-click and then choose Snap to Grid.

Copying Entity Relationship Diagrams

You can copy ERDs and paste them in other Microsoft applications, such as Word and Outlook.

NOTE: You cannot copy a drawing from one ERD and paste it into another ERD.

To copy an entity relationship diagram

1 Navigate to an entity relationship diagram.

For instructions, see "Navigating to Entity Relationship Diagrams" on page 76.

2 In the ERD canvas, right-click and then choose Copy > Drawing.

3 In a Microsoft application, such as Word or Outlook, choose Edit > Paste.

4 Configuring Tables and Columns

Topics in This Chapter

About Tables

Table object definitions are logical representations in the Siebel repository of the physical tables in the underlying database management system.

You can extend the Siebel data model using extension tables and extension columns to base tables but you cannot add new base tables, delete base tables and columns, or modify the properties of base columns.

NOTE: The term base table can refer to the table that an extension table extends, as specified in the Base Table property of the extension table's object definition. It can also refer to the table on which a business component is built, as specified in the Table property of the Business Component object definition.

Topics in This Section

Table Naming Conventions

Tables in the Siebel database use a three-part naming convention. The syntax is: PREFIX_NAME_SUFFIX.

PREFIX — Table names in Siebel eBusiness Applications have a one- to three-letter prefix (EIM_, S_, W_, and so on) to distinguish them from other tables in your application.

NAME — A unique table name that is generally an abbreviation of the entity supertype name.

SUFFIX — A supertype name may be followed by the entity subtype. For example, the supertype EVT (event) has ACT (activity) as one of its subtypes. Thus, the name becomes S_EVT_ACT.

The prefix indicates the part of the Siebel schema to which a table belongs. Table 11 provides some of the prefixes and their descriptions.

Table 11. Table Prefixes

Prefix	Meaning
EIM_	Interface tables for Enterprise Integration Manager.
S_	Siebel base table. (Exception: Tables with names of the form S_<*name*>_IF are obsolete interface tables.)
W_	Siebel eBusiness Data Warehouse table.

The suffix indicates a table type. Table 12 provides some of the suffixes and their descriptions.

Table 12. Base Table Suffixes

Suffix	Meaning
_ATT	File attachment table.
_REL	A table that supports a many-to-many relationship from an entity back to itself.

Table 12. Base Table Suffixes

Suffix	Meaning
_SS	A table that stores Siebel-to-Siebel integration information.
_X	One-to-one extension table, available for customers to add attributes to the Siebel database.
_XA	A table that stores extended attributes associated with an object class.
_XM	One-to-many extension table, available for customers to add attributes to the Siebel database.

Table Properties

The key properties of the table object type are listed below. For a complete list of Table properties, see *Object Types Reference*.

- **Name.** Provides the name of the table in the DBMS.
- **Type.** Defines the table type. See Table 13.
- **Base Table.** Identifies the base table if the table in the object definition is an extension table. If the table in the object definition is a base table, this property is blank. An extension table always identifies a base table.
- **User Name.** A longer, descriptive name that aids in identifying the table when used in configuration.
- **Comments.** Can be used to provide a long description of the table, such as the type of data that is stored in the table.
- **Status.** The current status of a table. States whether tables from previous versions of Siebel eBusiness applications can be used in the most recent version of the application.

Table 13 lists possible values for the Type property.

Table 13. Table Types

Type	Description
Data (Public)	Public data tables are among the original set of tables implemented in Siebel applications. They hold data that is made available through business components to developers and users. Public data tables can be extended using extension tables and extension columns.
Data (Private)	Private data tables are similar to public data tables, but cannot have extension columns.
Data (Intersection)	Intersection data tables implement a many-to-many relationship between two data tables.

Table 13. Table Types

Type	Description
Extension	Tables of type extension are additional tables with implicit 1:1 relationships to its parent base table, the table it logically extends. They provide additional columns that you can use to store data. Tables with implicit 1:M relationships to base tables are also commonly referred to as extension tables. However, these tables have a type property of Data (Public) not Extension.
Interface	Interface tables are used by Siebel Enterprise Integration Manager (EIM) to import data for populating one or more base tables and subsequently to perform periodic batch updates between Siebel applications and other enterprise applications. Interface table names end in _IF or _XMIF.
Database View	Reserved for Siebel internal use.
Dictionary	Reserved for Siebel internal use.
Journal	Reserved for Siebel internal use.
Log	Reserved for Siebel internal use.
Repository	Reserved for Siebel internal use.
Virtual Table	Reserved for Siebel internal use.
Warehouse Types	Reserved for Siebel internal use.
Extension (Siebel)	These tables are reserved for Siebel use only. They are usually extensions from S_PARTY.

About Extension Tables

Extension tables provide additional columns that you can use to store custom attributes. Extension tables have either an implicit 1:1 or a 1:M relationship with a given base table.

Siebel Systems provides a set of preconfigured extension tables that are set aside for customer use. These tables are known as static extension tables. Static extension tables include generic columns (ATTRIB_) that you can use to store custom attributes. Static extension tables are already part of the data model, so using them does not require updating the physical database.

NOTE: Some ATTRIB_ columns in extension tables are used by Siebel Systems. Do not modify or delete ATTRIB_ columns in use by standard Siebel applications.

You can also create your own extension tables using the New Table Wizard. Extension tables that you create are changes to the logical schema and therefore must be also be applied to the physical database.

When columns in a base table are updated, the timestamps of its extension tables are not updated unless columns in those extension tables are also updated. However, when records in an extension table are changed, system columns in a parent table are updated. This is done because the associated record in an extension table is considered by the object manager to be logically a part of its parent record.

Related Topics

"About One-to-One Extension Tables" on page 93

"About Implicit Joins" on page 94

"About One-to-Many Extension Tables" on page 96

"Using Static 1:1 Extension Tables" on page 112

"Using Static 1:M Extension Tables" on page 113

"Creating 1:1 Extension Tables Using the Object List Editor" on page 117

"Creating New Tables Using the New Table Wizard" on page 117

About One-to-One Extension Tables

Names of one-to-one extension tables are appended with the suffix _X. Rows in the extension table have one-to-one relationships with corresponding rows in the base table. They are basically an extension of the base table record. The Type property of a one-to-one extension table is set to the value Extension.

One-to-many extension tables require that you create new fields for the business component of the base table and map them to available columns in the one-to-one extension table. For example, consider the example shown in Figure 11. Three fields, Hobby, Married, and Spouse, are added to the Contact business component and map to columns in the S_CONTACT_X extension table.

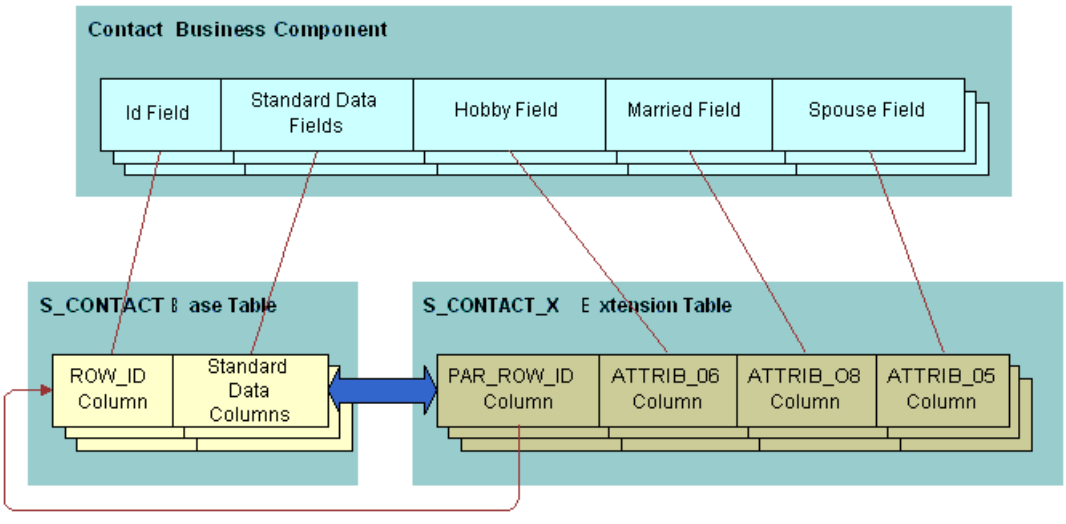


Figure 11. Extension Table Example

Related Topics

"Configuring Data-Driven Read-Only Behavior" on page 190

"Using Static 1:1 Extension Tables" on page 112

"Creating 1:1 Extension Tables Using the Object List Editor" on page 117

"Creating New Tables Using the New Table Wizard" on page 117

About Implicit Joins

Underlying a 1:1 extension table's relationship with the base table and business component is a set of hidden relationships called an *implicit join* (also called *implied join*). The implicit join makes the extension table rows available on a 1:1 basis to the business component that uses the extension table.

The name of the implicit join is the same name as the extension table. When a field in the business component is based on a column in the extension table, the Column property of the Field object is set to the name of the column, and the Join property is set to the name of the extension table. For example, the Industry field in the Contact business component has a Column property value of ATTRIB_48 and a Join property value of S_CONTACT_X.

An implicit join is different from a join that you define as an object definition.

- Implicit joins allow data to be updated in columns, whereas joins defined as object definitions do not. Joins defined as object definitions only allow data to be displayed.
- Implicit joins do not need to be configured. Columns in the extension table will automatically be available for you to use for business component fields.

The details of the object definition relationships in an implicit join are illustrated in Figure 12.

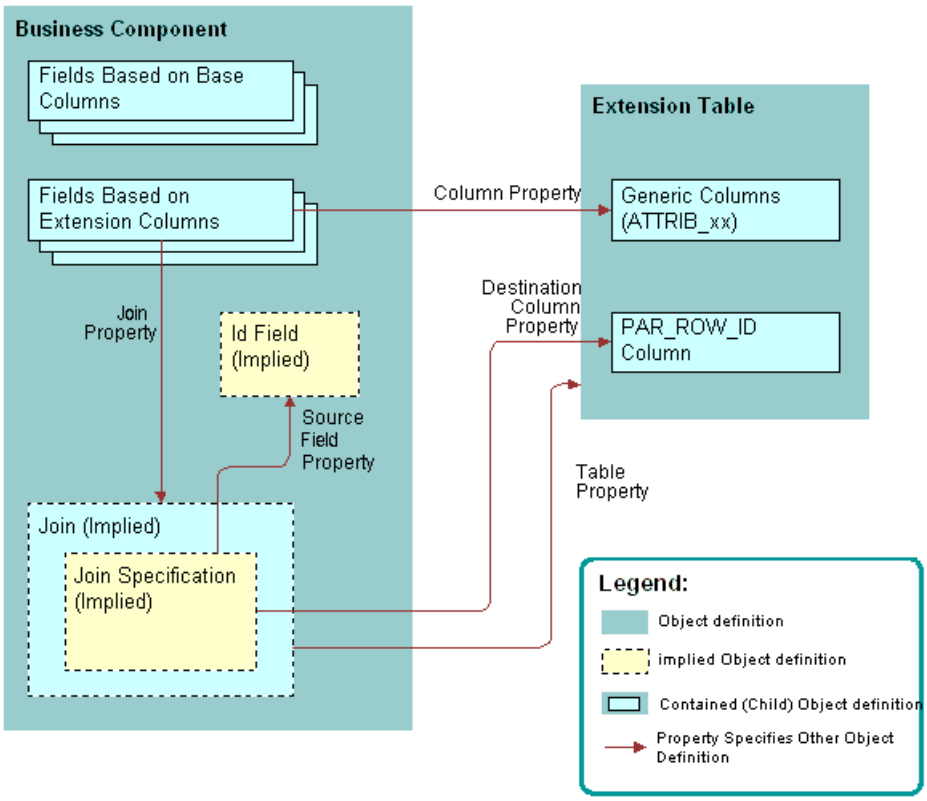


Figure 12. Extension Table Details with Implicit Join

The following definitions participate in the implementation of the implicit join:

- **Id field.** The Id field is a system field in the business component. It represents the ROW_ID column in the base table, and it can be used in joins involving extension tables and other joined tables.
- **PAR_ROW_ID column.** PAR_ROW_ID stands for parent row ID. Every extension table has this column, and every extension table row has a value there. It is used as a foreign key to the base table that is extended by the extension table.

Related Topics

"Using Static 1:1 Extension Tables" on page 112

"About Implicit Joins" on page 200

About One-to-Many Extension Tables

The names of 1:M extension tables are appended with the suffix _XM. In a 1:M extension table, there can be multiple rows for a single row in the base table. Like 1:1 extension tables, 1:M extension tables have a set of generic columns (ATTRIB_xx) that you can use to store custom attributes. However, unlike 1:1 extension tables, 1:M extension tables have a Type property value of Data (Public) rather than Extension.

You can use 1:M extension tables to track additional entities that have a 1:M relationship with a parent business component but are not represented by existing Siebel business components. You can store data for multiple business components in an _XM table. The Type column is used to group records in the _XM table. You configure each business component to retrieve only the rows of a given type.

One-to-many extension tables require the following configuration at the business object layer:

- A new business component and fields that map to columns in 1:M extension table that you want to use to store data.
- Three additional business component fields that provide the user key and map to the following columns:
  - PAR_ROW_ID. This column maps to the foreign key field using in the 1:M link.
  - NAME. The value stored in this column needs to make the record unique for each parent record.
  - TYPE. This column is used to group records in the extension table. Set a default value for the Type field and then configure the business component search specification to automatically search for those records in the extension table that contain the default value.

    NOTE: The combination of NAME, TYPE, and PAR_ROW_ID will be unique in order to satisfy the U1 index of the _XM table.
- A link and business object component to establish the master-detail relationship between the new (detail) business component and its master business component.

CAUTION: Avoid an _XM table as an extension to an existing _XM table. It will cause problems with EIM and docking processes.

Related Topics

"Using Static 1:M Extension Tables" on page 113

"Creating Business Components" on page 173

"Creating Links to Define a 1:M Relationship" on page 209

"Creating Business Objects and Business Object Components" on page 227

About Intersection Tables

An *intersection table* implements a many-to-many relationship between two business components. A *many-to-many relationship* is one in which there is a one-to-many relationship from either direction. For example, there is a many-to-many relationship between Opportunities and Contacts. One Opportunity can be associated with many Contact people, and one Contact person can be associated with many Opportunities. Two different views can appear (in different business objects) which associate the two business components in opposite ways, as illustrated in Figure 13 and Figure 14 on page 98.

Figure 13 shows the Account Detail - Contacts View, in which one account is displayed with multiple detail contacts.

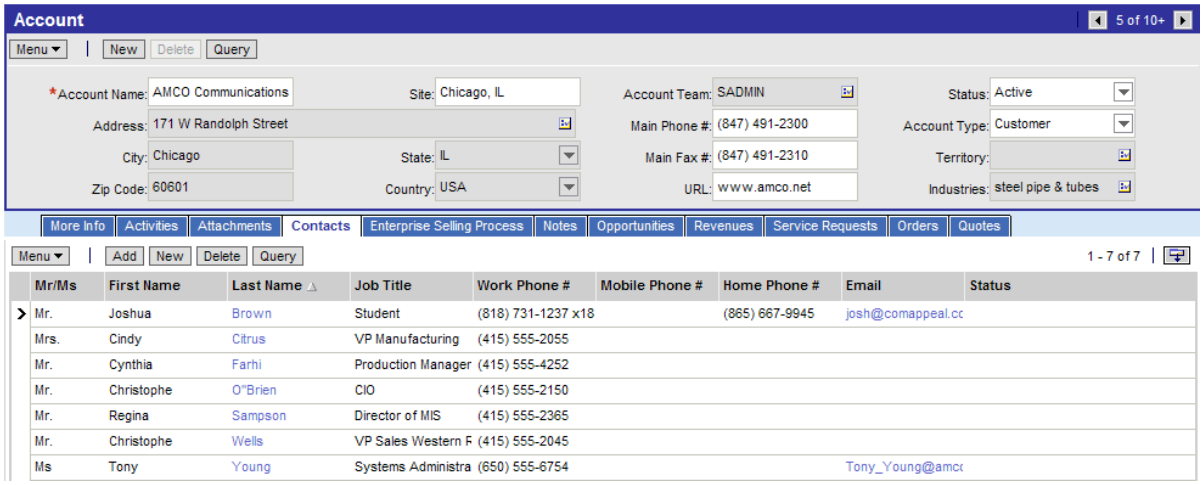


Figure 13. Account Detail - Contacts View

Figure 14 shows the Contact Detail - Accounts View, in which one master contact is displayed with multiple detail accounts.

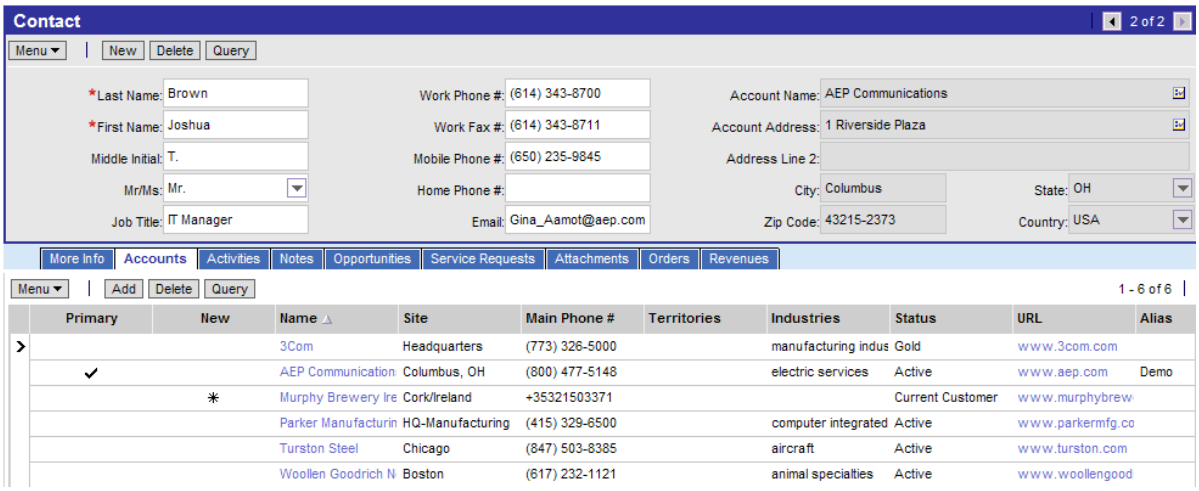


Figure 14. Contact Detail - Accounts View

To implement a many-to-many relationship, two links and a table designated as an *intersection table* are required. The table is designated as an intersection table in its Type property by means of a value of Data (Intersection). The intersection table represents the many-to-many relationship as two one-to-many relationships, which the underlying DBMS is designed to handle. There is no database construct that implements many-to-many relationships directly. This representation design is illustrated in Figure 15.

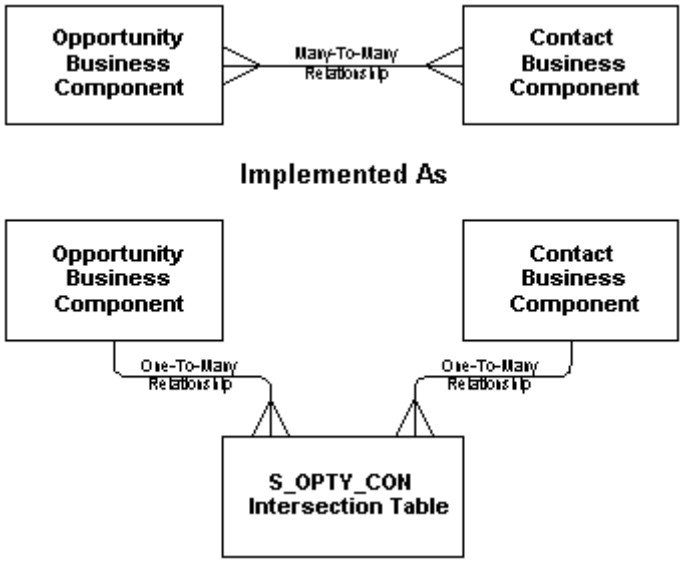


Figure 15. Many-to-Many Relationship as Two One-to-Many Relationships

For more information about links, see "About Links" on page 206.

You can configure custom intersection tables using New Table Wizard. For information, see "Creating New Tables Using the New Table Wizard" on page 117.

About How Intersection Tables are Used

The intersection table contains one row for each association between a row in one business component's base table and a row in the other business component's base table, regardless of which one-to-many relationship the association pertains to. The association row in the intersection table stores the ROW_ID values of the row in each business component base table. The details of intersection table relationships are illustrated in Figure 16.

Notice how the associations stored in one intersection table serve both the Opportunity/Contact and Contact/Opportunity links, and their corresponding views. An association is simply a pair of ROW_ID values pointing to rows in their respective business component base tables. One association may appear in both views, for example, the association between Cynthia Smith and Smith Dry Goods.

NOTE: The set of object definitions and relationships in Figure 16 on page 100 pertains to one of the two links. The other link uses the same set of object types, but slightly different relationships.

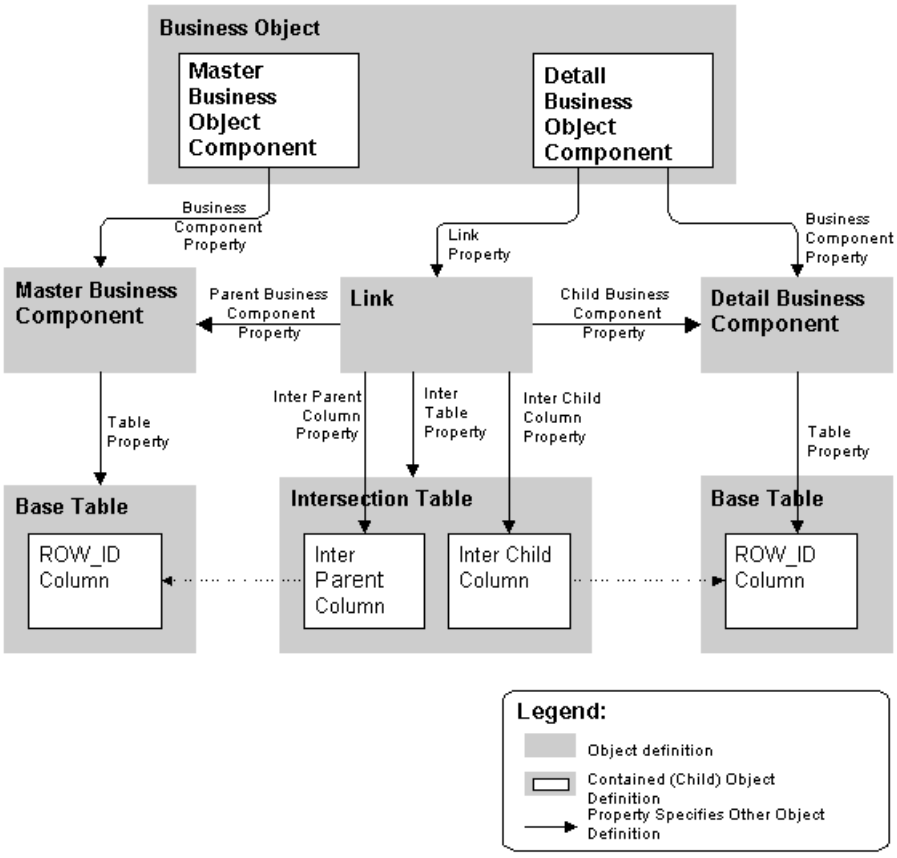


Figure 16. Intersection Table Details

The following are descriptions of the object definitions in Figure 16:

- **Business object.** The business object references the link (indirectly through the business object's child business object component) that uses the intersection table. It also contains the two business components included in the link.

- **Business object components.** Business Object Component object definitions are used to include business components in the business object. Business Object Component is a child object type of Business Object. The detail business object component references both the detail business component, by means of the Business Component property, and the link, by means of the Link property. The master business object component only references its corresponding business component.
- **Link.** The link object definition establishes a one-to-many relationship between the two business components in a particular direction. That is, the property settings in the link specify that one business component is the master and the other is the detail in the master-detail relationship.
- **Master and detail business components.** The two business components are specified in the link. They provide data to the user interface object definitions that display the master-detail relationship. The base table of each business component contains the ROW_ID column referenced by the Inter Child Column (detail) and Inter Parent Column (master) properties of the Link object type.
- **Intersection table.** The intersection table holds the associations between rows in the base tables of the master and detail business components. Each row in the intersection table represents one association between the two business components. Two columns in the intersection table serve as foreign keys to the base tables of the two business components. These columns are identified in the Inter Parent Column and Inter Child Column properties of the link.
- **Inter Parent column.** This column in the intersection table holds the pointer to the associated row in the master business component's base table. It is identified in the Inter Parent Column property of the Link object.
- **Inter Child column.** This column in the intersection table holds the pointer to the associated row in the detail business component's base table. It is identified in the Inter Child Column property of the Link object.
- **ROW_ID columns.** The base table of each business component has a unique identifier column for the rows in that table. This is the ROW_ID column.

NOTE: The Inter Table, Inter Parent Column, and Inter Child Column properties of the Link object type are specific to links used in implementing many-to-many relationships based on intersection tables, and are blank in other links.

Figure 17 illustrates the property settings in the two links used to implement a many-to-many relationship—in this case the relationship between Opportunities and Contacts. Notice how the inter child column of one link is the inter parent column of the other, and the other way around. Also notice how the parent business component in one link is the child business component in the other, and the other way around. The two links are mirror images of each other.

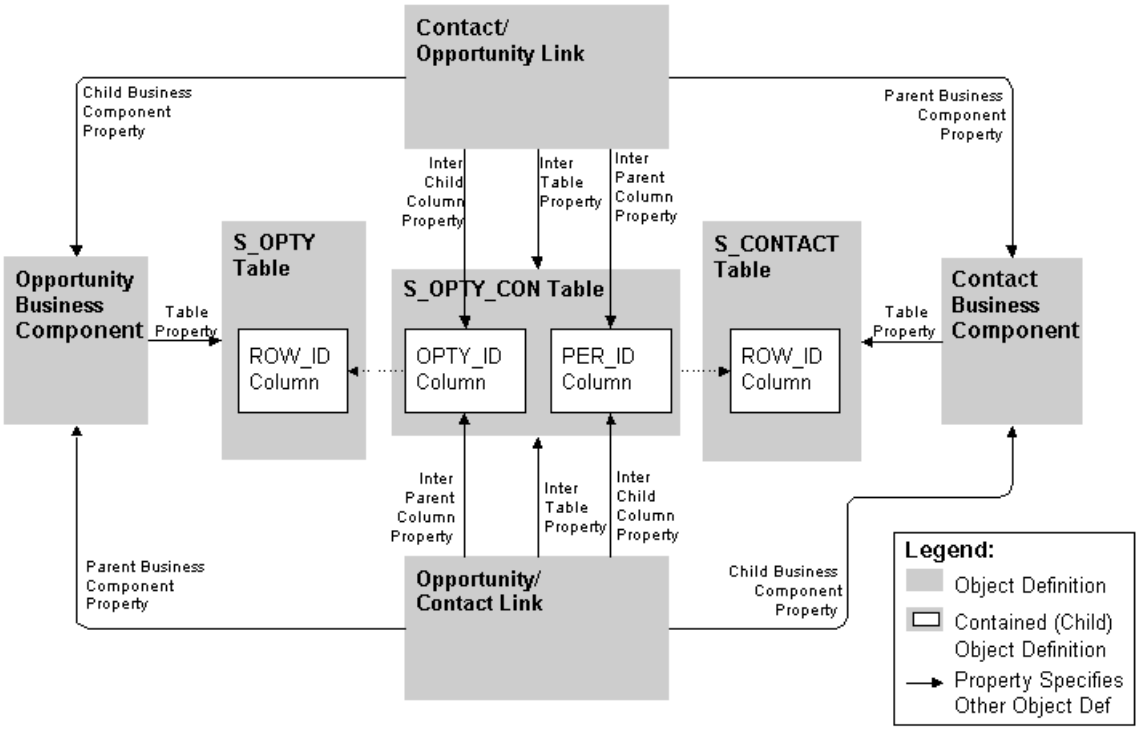


Figure 17. Two-Link Intersection Table Example

About Intersection Data in Intersection Tables

In addition to the two foreign key columns that establish relationships between the records in the two business components, an intersection table may contain various columns that hold data specific to the intersection of the two. These columns are called *intersection data columns*.

For example, in the S_OPTY_CON table, which implements the many-to-many relationship between Opportunity and Contact, there are several data columns in addition to OPTY_ID and PER_ID. These columns hold information about the combination of a particular opportunity and a particular contact. A description of a few of these columns follows:

- **ROLE_CD.** The role played by this contact in this opportunity.
- **TIME_SPENT_CD.** The time spent on this opportunity with this contact.
- **COMMENTS.** Comment specific to this combination of opportunity and contact.

Some intersection data columns are useful primarily to one master-detail relationship, some primarily to the other, and some to both. For example, ROLE_CD would make sense only in the context of a master-detail relationship in which an opportunity was the master record with multiple detail contact records. In contrast, TIME_SPENT_CD would make sense in the context of either master-detail relationship. That is, each contact has a unique role in the opportunity and the converse does not make sense. However, the time spent with each contact on an opportunity could be seen from the alternative perspective of the time spent on each opportunity with a contact.

An intersection data column is accessed by a field in a business component using a join. An implicit join exists for any intersection table, and has the same name as the intersection table. The implicit join is created when a link using an intersection table is created. It will exist for the child business component. For example, the ROLE_CD column in S_OPTY_CON is mapped into the Role field in the Contact business component. The Join property of this field has the value S_OPTY_CON. The Contact business component does not have a child join object definition named S_OPTY_CON; the join is automatically provided and invisible in the Object Explorer. This is similar to the implicit join that exists for one-to-one extension tables. Data can also be updated through the implicit join.

Intersection tables can be extended with extension columns. They cannot be extended with custom extension tables.

About Columns

Column object definitions are representations of the physical columns in the underlying database management system. The name, data type, length, primary key and foreign key status, alias, and other properties of the database column are recorded as properties in the corresponding column object definition. Additional properties internal to Siebel applications are provided in the object definition, such as the Changed and Inactive statuses, and Type (a classification for column object definitions).

Column Properties

The important properties of the Column object type are listed below. For a complete list of object properties, see *Object Types Reference*.

- **Name.** Provides the name of the database column in the database table.
- **Default.** Provides a default value when new rows of this table are added.
- **Physical Type (Physical Type Name).** Identifies the data type of the column in the database. The following data types are supported:
  - **Character.** Used for fixed-length text. Also used for Boolean columns, which are character columns with a length of 1. By default, you cannot have char greater than 1. To change the default setting, you need to set the preference in View > Options > Database.

    NOTE: Defining a Column as a Char when the data being stored in the column is variable in length will cause the data to be padded with blank spaces in the database. It is recommend that you use the varchar data type for all but Boolean columns implemented as CHAR(1).
  - **Long.** Long text. You can store approximately 16K worth of data in long columns.

  - **Varchar.** Variable-length text. Used for memo-type fields and to store row-ID and foreign key values. Used for most alphanumeric columns in the Siebel data model including ROW_ID, foreign key, LOV, and other free-form text columns.
  - **Number.** Any numeric data. Typical numeric columns in Siebel applications are 22,7 for general-purpose numbers, and 10,0 for integers.

    Data of this type is limited to 16 digits without a decimal point or 15 digits before a decimal point.
  - **UTC Date Time.** Time is saved in Greenwich mean time.
  - **Date.** Date values only, without time.
  - **Date Time.** Combined date and time values in the same column.
  - **Time.** Time values only, without the date.
- **Precision.** Specifies the maximum number of digits in a number column. For noninteger columns, the precision is 22. For integer columns, the precision is 10.
- **Scale.** Specifies the maximum number of digits after the decimal point. For noninteger columns, the scale is 7. For integer columns, the scale is 0.
- **Primary Key.** If TRUE, this column is the primary key for the table. With minor exceptions, the ROW_ID column in a table is the primary key, and has a TRUE value for this property.
  - **IFMGR:** xxx. These columns have names such as IFMGR: ROW_ID and IFMGR: Status. They are found in interface tables, and are for internal use by the Siebel Enterprise Integration Manager.
  - **System.** System columns appear in all tables in Siebel applications. However, no one set of system columns appears in every table. You can use the data in system columns for various purposes, although most system columns are read-only.
- **Nullable.** True/false value that specifies whether or not NULL can be stored in this column by the database.
- **Foreign Key Table.** Specifies the table to which this column is a foreign key, used by EIM. Leave NULL in extension columns.
- **User Key Sequence.** The sequence in the user key where this column fits.
- **LOV Bounded.** A TRUE or FALSE value. If LOV Bounded is TRUE, EIM will check the values as it imports against the values contained in a list defined in LOV Type. In that case, LOV data should be imported first into S_LST_OF_VAL, and LOV Type must be specified. This property is read-only for standard columns in Siebel applications but is editable for custom extension columns.
- **LOV Type.** Specifies the list of values domain in which this column is validated. Used in conjunction with the LOV Bounded property. List of values domains are defined in List of Values Administration in the client. This property is read-only for standard columns in Siebel applications but is editable for custom extension columns.

About Data Columns

Data columns comprise most of the columns in Siebel applications. They are sometimes referred to as base columns. Data columns provide the data for fields, or serve as foreign keys that point to rows in other tables. You cannot modify the properties of data columns. Data columns can be public or private.

About Extension Columns

An extension column is used to store custom attributes. There are three kinds of extension columns:

- **Static extension columns.** Extension columns are included in static extension tables for customer use. They are named ATTRIB_*nn*, where *nn* is a value between 01 and 47 (for example, ATTRIB_13). Modifying or deleting static extension columns is not recommended.
- **Custom extension columns in an extension table.** These are columns added by the developer to an extension table. They have the prefix X_ in their names.
- **Custom extension columns in a base table.** These are columns added by the developer to a base table. The relational database system that you use with Siebel applications determines whether or not this is allowed. When the database system supports custom extension columns in base tables, it may be preferable for performance reasons to add them there, rather than to an extension table. Performance may be affected if the extension columns are added to an extension table, because extra SQL is generated to join to the extension table.

About System Columns

System columns appear in all tables in Siebel applications, although the same set of system columns does not appear in every table. You can use the data in system columns for various purposes; for example, the ROW_ID column in tables is used in the construction of joins. Generally you should not modify the data in system columns. However, there are exceptions, such as certain system columns in interface tables. Some common system columns are described below:

- **ROW_ID.** The ROW_ID column is present in all tables and provides a unique identifier to the rows in the table. It is the typical destination column of foreign key relationships from other tables. In standard data tables, it is often represented by a field called Id for use in joins and links. For example, the ROW_ID column in the S_ORG_EXT table is represented as the Id field in the Account business component.

  NOTE: The Id field that represents the ROW_ID column in business components is an implicit field, and does not appear in the Object Explorer as a child field of any business components. However, every business component has an Id field, which represents the ROW_ID column of its base table, as defined in the Table property of the business component. The Id field is referenced in various property settings throughout Siebel applications, such as in the Source Field property of a link (in which a blank value also means the Id field).

- **CREATED.** Provides the creation date and time of each record.

- **CREATED_BY.** Stores the ROW_ID of the S_USER record of the person who created the record—not to be confused with the user name that the user logged in with.
- **LAST_UPD.** Provides the date of last update of each record.
- **LAST_UPD_BY.** Stores the ROW_ID of the S_USER record of the person who last updated the record—not to be confused with the user name that the user logged in with.

  NOTE: The CREATED, CREATED_BY, LAST_UPD, and LAST_UPD_BY columns show the client date-time and user values. They do not show server DBMS date-time and user values.

- **PAR_ROW_ID.** The PAR_ROW_ID column is a foreign key to the ROW_ID column of the base table. Extension tables, as well as _ATT and _T tables, have this system column.

LAST_UPD, ROW_ID, LAST_UPD_BY, CREATED, and CREATED_BY columns are system fields that are updated automatically by the Siebel application.

System fields should not be explicitly defined for a business component. If the business component fields that are based on these columns are defined, the application will attempt to write a value to these columns twice in the insert statement and will cause a duplicate column SQL error.

Table 14 identifies the correspondences between system fields and system columns.

Table 14. System Fields and Their System Columns

System Field Name	System Column Name	Description
Id (or blank)	ROW_ID	Primary key for the table.
Created	CREATED	Creation date and time of the row.
Created By	CREATED_BY	Stores the ROW_ID of the S_USER record of the person who last updated the record.
Updated	LAST_UPD	Date of last update of the row.
Updated By	LAST_UPD_BY	Stores the ROW_ID of the S_USER record of the person who last updated the record. In some cases, this field is updated even though the user does not actively update the record. For example, this may occur when a multi-value link is configured with a primary join. See "Configuring the Primary ID Field" on page 219 for more information.

These fields are automatically provided, and do not need to be explicitly declared. You can reference them in the Field property of controls, list columns and other object definitions, even though they do not display in the Object List Editor for the business component.

About Indexes and Index Columns

An index object definition is a logical representations of physical index in the underlying database management system. Siebel applications include a set of standard indexes and are named with a prefix of S_.

You cannot modify or delete standard indexes; however, if an index would benefit your implementation, you can create custom indexes.

The Index object type has the following key properties:

- **Name.** The name of the database index.
- **Unique.** A TRUE or FALSE value indicating whether multiple rows with the same value are allowed.
- **Type.** Indicates which of the following styles describes the index:
  - **Primary Key value.** A primary key index is indexed on the ROW_ID column.
  - **User Key value.** A user key index is developer-created. The set of index columns is developer-specified. It must consist of a unique combination of columns.
  - **Extension value.** An extension index is created by default when the developer adds an index. The set of index columns is system-specified.
  - **System value.** System indexes are included in standard Siebel applications, and you cannot modify them.

About Index Column Object Type

Index Column is a child object type of the Index object. An Index Column object definition associates one column to the index that is the parent object definition of the index column.

The Index Column object type has the following important properties:

- **Column Name.** The name of the column object definition to include in this index object definition.
- **Sequence.** The integer value that indicates the order of the column in the index relative to other columns. The Sequence property must be populated even when only one column is present.
- **Sort Order.** The sort order for the index column. Its value can be either Asc (ascending) or Desc (descending).

About User Keys

A user key specifies columns that must contain unique sets of values. It is used to determine the uniqueness of records during data import operations in EIM. The purpose of user keys is to prevent users from entering duplicate records.

A user key is designated by the name of its parent table with an _U*n* suffix, such as S_PROD_INT_U1. Each user key has User Key Column child objects that specify the table columns that must have unique values, for example BU_ID, NAME, and VENDR_OU_ID in the S_PROD_INT_U1 user key.

A predefined index exists for each Siebel-defined user key. This index also takes the form S_*TABLE_NAME*_U*n*.

NOTE: Do not modify user keys in standard Siebel tables or EIM base tables. This is a restricted activity and should not be attempted without assistance.

For more information, see "About Interface Tables" on page 129.

About the S_Party Table

The party model organizes entities such as Person, Organization, Position, and Household. A party always represents a single person or a group that can be translated into a set of people such as a company or a household. Siebel data access technology makes use of the Party model. Certain parts of the data model use the Party model to abstract the difference between people, companies, households and other legal entities. This covers relationships between your company and people (contacts, employees, partner employees, users) and other businesses (accounts, divisions, organizations, partners). The base table for all such access is S_PARTY. Related tables are implicitly joined as extension tables. Table 15 lists the extension tables and their corresponding EIM interface tables.

Table 15. S_PARTY Extension Tables and Corresponding EIM Interface Tables

Data Type	Extension Table to S_PARTY	EIM Interface Table
Accounts	S_ORG_EXT	EIM_ACCOUNT
Business Units	S_BU	EIM_BU
Contacts	S_CONTACT	EIM_CONTACT
Employees	S_CONTACT	EIM_EMPLOYEE
Households	S_ORG_GROUP	EIM_GROUP
Positions	S_POSTN	EIM_POSITION
Users	S_USER	EIM_USER

Because these extension tables are implicitly joined to S_PARTY, they are available through S_PARTY. Two preconfigured intersection tables (S_PARTY_PER and S_PARTY_REL) implement M:M relationships between party business components, such as Account and Contact. Which one you use depends on whether or not you need to enforce access control.

Use S_PARTY_PER when implementing M:M relationships between two party business components where you need to define access control. Records in S_PARTY_PER propagate data access rights from the parent to the child parties. However, it is important to minimize the number of rows in S_PARTY_PER to maintain good response times in visibility-constrained queries. Therefore, when implementing M:M relationships where you do not need to enforce access control, such as when implementing a recursive M:M relationship between a party business component and itself, use S_PARTY_REL.

For example, use S_PARTY_PER to implement relationships between members:

- Access groups and members
- Accounts and contacts
- Employees and positions
- User lists and users

If you need to extend tables in the party model, you need to create an extension table from S_PARTY. For example, S_CONTACT is an extension table of S_PARTY. Because S_CONTACT is of type Extension (Siebel), you cannot use it as a base table for an extension table. You must create an extension table and use S_PARTY as the base table. To display data from the new extension table, create a Join object (explicit join) to bring in data from the new extension table to the business component you are using.

About Database Extension Options

When you need to store data that is not part of the standard Siebel data model, you have several options, including using static (preconfigured) extension tables and columns, adding columns to existing base tables, and creating new extension tables. Options for extending the data model are grouped into Standard and Advanced Database Extensibility.

Standard Database Extensibility

Standard database extensibility provides the following options:

- **Using Static Extension Tables and Columns.** These are predefined tables and columns that are available for you to use for your own purposes. They are the easiest option for storing additional entities because they are already part of the data model, so using them does not require you to modify the logical schema. Static extensions include: extension Columns, 1:1 Extension Tables, and 1:M Extension Tables. For more information, see "Using Static 1:1 Extension Tables" on page 112 and "Using Static 1:M Extension Tables" on page 113.
- **Adding extension columns to base tables.** You can use the Database Designer, available in the Object List Editor, to add columns to base tables. For more information, see "Adding Extension Columns to Base Tables" on page 114.
- **Creating New 1:1 Extension Tables.** You can use the Database Designer, available in the Object List Editor, to create new 1:1 extension tables. For more information, see "Creating 1:1 Extension Tables Using the Object List Editor" on page 117.

Advanced Database Extensibility

Advanced database extensibility provides the following options:

- **Creating New Tables Using the New Table Wizard.** You can extend the data model by creating the following types of tables:
  - Stand-alone tables.
  - 1:1 Extension Tables.
  - 1:M Extension Tables.

- Intersection tables.

For more information, see "Creating New Tables Using the New Table Wizard" on page 117.

- **Mapping Extensions to interface tables.** The EIM Table Mapping Wizard Allows you to create or associate the new table to the appropriate interface tables for using EIM. You can generate EIM Table Mapping objects for importing data into tables you have created, and you can automate the creation of EIM Attribute maps on extension columns added to base tables. For more information, see Chapter 5, "Configuring EIM Interfaces."
- **Mapping Extensions to Dock Objects.** The Dock Object Mapping Wizard allows you to associate the new table with an existing or new customer Dock object to support the synchronization of its data to remote users. For more information, see Chapter 6, "Configuring Docking Rules."

Guidelines for Extending the Data Model

When extending the database consider the following:

- Do not modify standard base tables and their columns.
- In most cases, try to use static extension tables and columns to meet design requirements. They are preconfigured and already part of the Siebel data model so they do not require modification of the schema or the physical database. If static extension tables and columns are not available, then explore other options, such as creating new extension tables.
- When deciding whether to add an extension column to a base table or to use columns in an extension table, consider the following recommendations:
  - Add extension columns to base tables when the data you need to store almost always exists for a given base record and it is accessed regularly. By avoiding the join used with an extension table, this approach often results in better performance. However, note that it can result in slower access to the base table if there is a lot of data (that is, many large fields have been added and they are always populated), because fewer rows now fit on one page.

    NOTE: When an extension column will be regularly specified in user queries, it is likely to need an index on the column that may need to include other base table columns; and therefore should be added to the base table.

  - Use columns in an extension table when a one-to-many extension fields are required, and the view displaying this data is accessed infrequently. In this case, the join is executed for the extension table, but only when this view is accessed.
  - As a general rule, if you plan to display additional attributes in a list applet, add extension columns to the base table; if you plan to display additional attributes in a form applet, use extension table columns.
- The standard user interface does not use all the relationships available in the underlying data model; however, most entity relationships are available for you to use. During the discovery phase of an implementation, try to meet business requirements using the existing data model and standard objects.

- To minimize the impact of your changes on other developers, make any bulk changes to the schema at the beginning of each project phase. If you make changes during a project phase, then these need to be distributed to all mobile users. You can use Siebel Anywhere to distribute a schema change; otherwise, you generate new database extracts for all of your mobile users before you can progress to the next phase.
- Use caution when configuring a extension column to hold foreign keys. Foreign key extension columns may be appropriate when pointing to enterprise-visible business objects, but they should be avoided when pointing to limited-visibility business objects such as Opportunity, Contact, Account, or Service Request. Using extension columns foreign key columns can cause problems when generating EIM mappings and when routing data to mobile users.

  You cannot configure EIM to import data into a foreign key column, because the necessary EIM object types cannot be configured by customers.
- Do not define columns with names longer than 18 characters in the DB2 environment.
- You cannot add EIM mappings for foreign key relationships to tables that do not have user key attributes.
- The Siebel Data Model consists of over 2,000 database tables. Each of these tables follows a standard naming convention to help users identify individual tables. For information on naming conventions for tables, see "About Tables" on page 89.
- When planning or implementing Mobile Web Client users, be aware that downloading data to the local database is governed by Dock Object Visibility rules. These rules use the standard relationships to determine which tables' data are routed to the mobile user's local database.

  Thus, when new relationships are created, there are no Dock Object Visibility rules that allow relevant data to be downloaded to the local database. This may cause users to not be able to see their data.

  To resolve this, you can use a Docking Wizard feature to create custom docking rules for custom foreign keys. However, you may encounter some performance concerns if you do not analyze the results of the functionality implemented by the Docking Wizard before you implement a new Dock Object Visibility rule or object. Primarily, the performance of your remote processes (such as the Transaction Processor and Router) may be affected.

  In addition, by adding a rule you may be inadvertently adding a significant number of database records to remote users, which could affect initialization and synchronization times. An increased number of records in the remote database may also impact the mobile user application's performance.

  For more information about Dock Objects, see "Configuring Docking Rules" on page 149.

Using Static 1:1 Extension Tables

Siebel applications provide 1:1 static extension tables for many of the standard data tables. The static extension table contains columns of various types that are set aside for your use. It has a predefined 1:1 relationship with a base table that allows you to use the additional columns in the extension table for new functionality without making alterations to the base table or modify the database schema. When using static 1:1 extension tables, you do not need to create a new business component object because these tables are already implicitly defined as joins. For more information about implicit joins, see "About Extension Tables" on page 92.

1:1 Static extension tables have a suffix of _X in the name, such as S_PROD_INT_X. Columns in these tables are named ATTRIB_*nn*, where *nn* is a value from 01 to 47. You can add additional extension columns if needed.

Not all columns in standard extension tables are available to use because Siebel applications use some of the columns. Extension columns used by standard Siebel applications should be treated as data columns in base tables—that is, they should not be modified or deleted.

Table 16 lists the different data types found in Siebel extension tables and the number of columns of each data type.

Table 16. Standard Extension Columns

Data Type	Number of Columns
Number	12
Date	10
VarChar(255) VarChar(100)	1 5
VarChar(50)	10
VarChar(30)	5
Char(1)	4

Before deciding to use a static extension table, you need to see if the column that you want to use in the table is already being used by standard Siebel configurations. You do this by searching the repository for fields that are associated with the column that you want to use.

CAUTION: If the column is in use by a field defined by a Siebel application, do not deactivate the original field in order to use that column for another purpose.

To determine if standard extension columns are available

1. In the Siebel Tools Object Explorer, click the Flat tab.
2. Select the Field object type.
3. Choose Query > New Query.
4. In the Object List Editor, enter the following search criteria:

- In the Column property, enter the name of the column you want to use.
- In the Join property, enter the name of the extension table.

If the query does not return any Field object definitions, the column is unused in the extension table and is available. If the query returns one or more object definitions, find another extension column in that table. To determine which extension columns are currently in use, perform the query again with the same extension table specified (in the Join property) and the value "ATTRIB*" in the Column property.

Using Static 1:M Extension Tables

There are more than 20 predefined tables that have one-to-many relationships with base tables. These tables have the suffix _XM. They have generic columns that you can use to store additional data. They allow you to track entities that do not exist in standard Siebel applications and have a 1:M relationship to an existing base table. Because the extension tables themselves are already part of the data model, you do not need to modify the database schema.

For information on how to display data from static 1:M extension tables, see "About Extension Tables" on page 92.

Process for Extending the Data Model

When extending the data model, perform the following tasks:

1. Check out projects that contain the tables you want to modify.

   For information about checking out of projects, see *Using Siebel Tools*.

2. Make the requisite changes to the logical schema, such as creating tables, columns, or indexes. Tasks may include:
   - "Adding Extension Columns to Base Tables" on page 114
   - "Creating Columns of Type LONG" on page 115
   - "Creating 1:1 Extension Tables Using the Object List Editor" on page 117
   - "Creating New Tables Using the New Table Wizard" on page 117
   - "Modifying Extension Tables or Columns" on page 121
   - "Deleting Extension Tables or Columns" on page 122
   - "Creating Custom Indexes" on page 123
3. "Applying Database Extensions to the Local Database" on page 124
4. "Preparing Server Database for Applying Schema Changes" on page 125
5. "Applying Data Model Changes to the Server Database" on page 126
6. "Propagating Changes to Other Local Databases" on page 128

Adding Extension Columns to Base Tables

You can add extension columns to existing base tables. Adding an extension column avoids additional joins to extension tables for storing custom attributes. You can add extension columns to any of the following table types:

- Data tables
- Intersection tables
- Interface tables
- Standard extension tables
- Custom extension tables
- Extension (Siebel)

You cannot add columns to tables in the following cases:

- You cannot add extension columns to private data tables—that is, tables with a Type property of Data (Private).
- Some interface tables are private, although most are public.

When adding columns to tables, consider the following:

- Review "Guidelines for Extending the Data Model" on page 110.
- Any columns you add must conform to the data type limitations of all the DBMS types used in your Siebel enterprise. Consider your server database as well as any regional or mobile databases.
- For Oracle databases, the maximum length of extension columns with data type of Varchar is 2000 characters. If you create a column of type Varchar that is longer than 2000, it is automatically implemented as a column with data type of LONG.
- If you add new mandatory columns to an existing table with one or more rows of data, the DBMS will not allow the column to be added unless a default value is provided.
- After a column has been added to the physical table, a column cannot be removed. Refer to your database system's documentation for information on this constraint.
- When you create a new extension column in the Siebel schema, there might be padding issues with Siebel Remote. For more information about Siebel Remote, see *Siebel Remote and Replication Manager Administration Guide*.

- When you add columns to tables, do not use column names that are reserved words on your server or client database. However, you can use reserved words if you include an underscore ("_") somewhere in the column name. For more information about naming conventions, see "About Tables" on page 89.

CAUTION: Be extremely careful when using custom extension columns to track foreign keys. If you choose to implement this, it is recommended that you consult with Siebel Systems concerning the visibility rules applied to the foreign key table. Additionally, you must set the Foreign Key Table Name property to NULL for that column to load values into it using Enterprise Integration Manager. For information on creating foreign key mappings for EIM, see "About Interface Tables" on page 129.

To add an extension column

1. Connect to your local development database with Siebel Tools.
2. In the Object Explorer, expand the Table object type.
3. In the Object List Editor, select the table to which you will add an extension column or columns.
4. Make sure that the table is not of type Data (Private).
5. Select Column in the Object Explorer.
6. Select Edit > New Record to add an extension column.

   The logical database schema is changed based on the information you entered, but you still need to physically apply the changes to your local database as described in "Select Edit > New Record to specify each column to add to the index." on page 124.

Creating Columns of Type LONG

You can create an extension column of type LONG, but you are limited to one column of this type per table.

When configuring LONG columns, consider the following:

- You can only add LONG columns to 1:1 Extension Tables that have a valid base table identified in the Base Table property of the table object definition.
- You cannot add LONG columns to _XM tables because they are of type Data (Public).
- You cannot add LONG columns to tables of type Data (Public), such as S_EVT_ACT. Siebel Systems reserves the right to implement LONG columns in these tables.
- You can use LONG columns to store a maximum of 16 KB or 16383 characters.
- Querying a LONG column triggers additional I/O in your DBMS that is not necessary with other column datatypes. This extra I/O slightly increases the time required to retrieve each row of data from the database, which can add up to a noticeable reduction in performance when retrieving many rows of data from the database.

To create a LONG extension column

1. Find an appropriate 1:1 extension table that corresponds to the base table that needs the LONG column available.

   An example of a 1:1 extension table for S_EVT_ACT, for instance, is S_EVT_ACT_X.

2. Create a column in the table and set the Physical Type to 'Long' and Length set to '0'.

   For instructions on adding columns to tables, see "Adding Extension Columns to Base Tables" on page 114.

   The logical database schema is changed based on the information you entered, but you still need to physically apply the changes to your local database as described in "Select Edit > New Record to specify each column to add to the index." on page 124.

Creating 1:1 Extension Tables Using the Object List Editor

In cases where standard extension tables and columns are not available, you can create custom 1:1 extension tables.

When creating custom 1:1 extension tables, consider the following:

- Review "Guidelines for Extending the Data Model" on page 110.
- If you need to extend tables whose type is Extension or Extension (Siebel), you must extend from from the table's base table, not from the extension table itself. The Base Table property of the extension table will tell you which base table to extend. For example, S_CONTACT is an extension table of S_PARTY. Because S_CONTACT is of the type Extension (Siebel), you cannot use it as a parent table for an extension table. Instead, extend S_PARTY and use a join (implicit join) to display the data from the extension table.
- Custom 1:1 extension tables do not need new docking rules, as the data contained in these tables is implicitly routed according to the docking rules of their parent tables.

To create a custom 1:1 extension table

1. Connect to your local development database with Siebel Tools.
2. In the Object Explorer, select the Table object type.
3. In the Object List Editor, select the base table for which you want to create an extension table. Verify that its Type property has a value of Data (Public).
4. Click Extend.

   The extension table appears in the list of tables in the Object List Editor. The Database Extension Designer automatically creates the necessary standard columns and standard indexes. Temporary columns in interface tables that are imported to the base table for this extension table are also created automatically.
5. Create any additional extension columns on the custom extension table, following the instructions in "Adding Extension Columns to Base Tables."

Creating New Tables Using the New Table Wizard

The Table Wizard allows you to create new stand-alone tables, extension tables, and intersection tables. It provides picklists with appropriate choices for each type of table and makes sure that the naming conventions are observed.

When creating new tables, consider the following:

- You can only create tables of types: Data (Public), Data (Intersection), and Extension.
- You must explicitly grant permissions on any table that you create.

- Use the New Table Wizard only after exploring other ways of meeting your business requirements, such as using static extension tables or repurposing existing tables.
- Review "Guidelines for Extending the Data Model" on page 110.

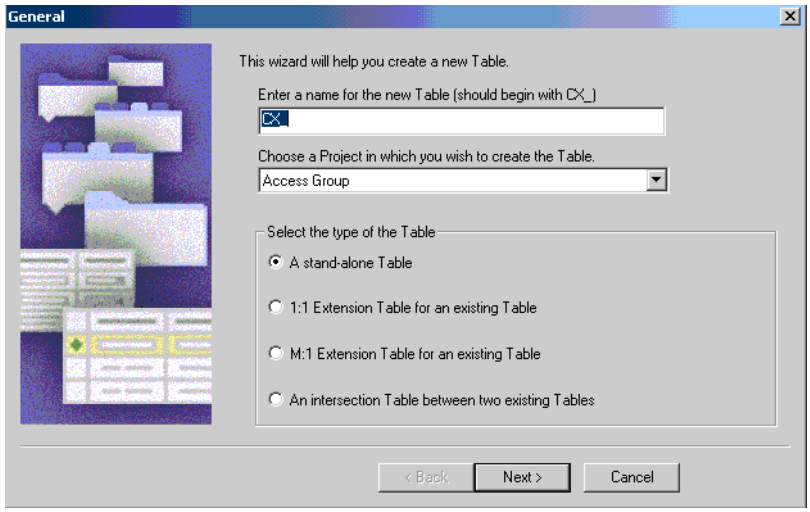
To use the Table Wizard

1 Choose File > New Objects.

The New Objects dialog box appears.

2 Select the Table Wizard icon.

The General dialog box appears.



3 In the "Enter a name for the new Table" field, it is noted that you must enter a new table that starts with "CX_" or it will automatically add a prefix.

NOTE: The Table Name must be uppercase. Mixed case or lowercase names may lead to problems when applying the changes on certain databases.
If you choose 1:1 extension tables, "_X" is suffixed to the table names.

4 In the "Choose a Project in which you wish to create the Table" field, choose a project.

NOTE: The Project list is restricted only to those projects that have been locked by the developer. All picklists are restricted to objects that belong to projects that are locked.

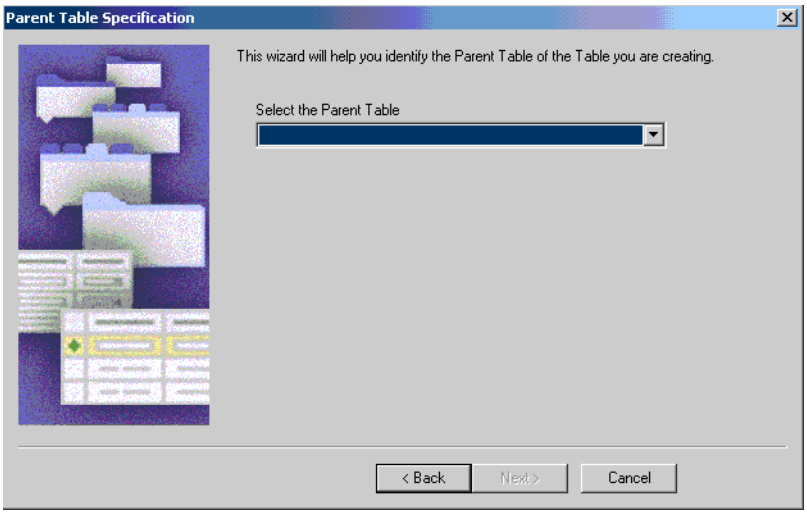
5 In the "Select the type of the Table" field, choose from the options: A stand-alone Table, 1:1 Extension Table for an existing Table, M:1 Extension Table for an existing Table, An intersection Table between two existing Tables.

6 Click Next.

NOTE: The next dialog box displayed depends on the type of table that is being added.

a If you choose Stand-alone Tables, the Parent Specification Table dialog box is not displayed and you are taken to the Finish dialog box, stating that the new table can now be created.

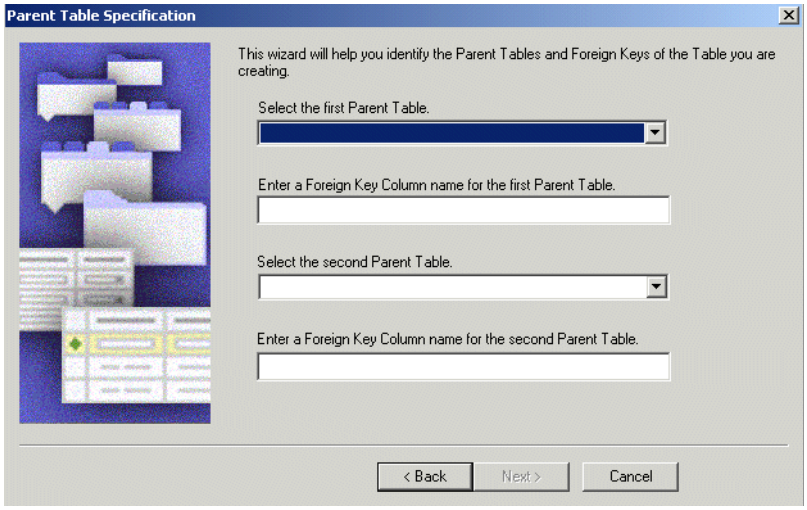
b For 1:1 Extension Tables and M:1 Tables, the Parent Specification Table dialog box allows you to select the parent table. See the following figure:



c For 1:1 Extension Tables, the picklist of available parent tables is restricted to tables of type Data (Public).

d For M:1 Extension Tables, the picklist of available parent tables is restricted to tables of type Data (Public).

e For Intersection Tables, the dialog box allows you to add both parent tables and names of foreign key columns to the parent tables. See the following figure.



- The picklist for the "Select the first Parent Table" field is restricted to all tables of type Data (Public).
- The picklist for the "Select the second Parent Table" field is restricted to tables of type Data (Public) with the following added restrictions, based on the choice of the first parent table.

- The names of the Foreign Key columns ("Enter a Foreign Key Column name for the first Parent Table" field and "Enter a Foreign Key Column name for the second Parent Table" field) are verified to make sure that they are unique (that is, do not conflict with each other or the system column names).

7 Click Next on the Parent Table Specification dialog box.

The Finish dialog box appears, which allows you to review the changes made before the objects are actually created. The Finish dialog box verifies that "The new Table can now be created" and asks you to make sure that the information about the Name, Project, Type of Table, and Parent Table 1 is correct.

8 Click Finish to generate the table.

You then see the new table listed (User Name CX_X) in the Object List Editor (the Type is Extension, the Base Table is CX).

Table Wizard Actions

The following columns are generated by the Table Wizard.

- For all types of tables, the Table Wizard creates seven system columns and the P1 index on ROW_ID.
- For 1:1 Extension Tables, the Table Wizard sets Type of Table = Extension and creates the following:
  - PAR_ROW_ID column
    - User Key Sequence=1
    - Foreign Key Table=<Base Table Name>
  - U1 index comprised of PAR_ROW_ID(1) and CONFLICT_ID(2)
    - Unique/Cluster=TRUE
    - Type=User Key
    - User Primary Key=TRUE
- For M:1 Extension Tables, the Table Wizard sets Type of Table=Data (Public) and creates the following:
  - PAR_ROW_ID, TYPE, NAME columns
  - U1 index comprised of PAR_ROW_ID(1), TYPE (2), NAME (3), and CONFLICT_ID (4)
    - Unique/Cluster=TRUE
    - Type=User Key
    - User Primary Key=TRUE
  - M1 index on TYPE (1) and NAME (2)
    - Unique/Cluster=FALSE
    - Type=System

- For intersection tables, the Table Wizard sets the type of the table to Data(Intersection) and creates the following:
  - TYPE column for added user functionality
  - Two Foreign Key columns with names specified in the Table Wizard

    User Key Sequence=1 and 2

    Foreign Key Table=<Parent Table>
  - U1 index on the two Foreign Keys (1, 2), TYPE (3), and CONFLICT_ID (4)

    Unique/Cluster=TRUE

    Type=User Key

    User Primary Key=TRUE
  - F1 index on the Foreign Key to the second parent table

NOTE: When a custom extension table is added using the Table Wizard, a U1 index is added to the table. However, the User Key column is blank and does not allow the definition of user keys. This is because there is no need to create user keys: they are only needed to resolve foreign keys while using EIM, and EIM does not work with foreign keys to custom tables.

Modifying Extension Tables or Columns

You may need to modify your extension tables or columns after creating or activating them. You can modify properties of extension tables or columns only. Modification of standard base tables and their columns is not supported.

You can rename a column prior to applying it to the server. However, once the column has been added or applied to the server, you cannot simply rename the column and must deactivate the existing column and create a replacement extension column.

NOTE: You cannot modify the static extension tables that are preconfigured with Siebel applications. See "About Extension Tables" on page 92.

To modify an existing extension column

1. Connect to your local development database with Siebel Tools.
2. In the Object Explorer, expand the Table object type.
3. In the Object List Editor, select the table containing the extension column to modify.

   NOTE: If you are adding a new extension table to the EIM Table Mapping list, make sure you click Activate to create all the necessary temporary columns that are needed for EIM processing.
4. In the Object Explorer, select Column.
5. In the Object List Editor, select the extension column to modify.
6. Modify the appropriate properties.

To rename an existing extension column

1. Connect to your local development database with Siebel Tools.
2. Navigate to the table you want to modify, and then deactivate the unwanted column.
3. Create a new column in Siebel Tools (the logical schema).
4. Export the data from the old column.
5. Drop the old column from the table.
6. Use ddlsync.ksh to synchronize the logical and physical schema and import the data back in.

To modify an existing extension table

1. Connect to your local development database with Siebel Tools.
2. In the Object Explorer, select the Table object type.
3. In the Object List Editor, select the extension table to modify.
4. Modify the appropriate properties.

NOTE: Be careful when modifying the Physical Type property for columns; depending on existing data in the column, changes may not be possible.

Deleting Extension Tables or Columns

You can delete extension tables and columns that you have created from the logical schema. Deleting tables and columns removes them from the logical schema in the Siebel repository, however it does not remove them from the physical schema of the database.

NOTE: You can delete only custom extension columns and tables; you cannot delete any standard tables or their columns.

After you delete an extension table, any corresponding temporary columns in an interface table are not deleted. These columns cannot be deleted through Siebel Tools, and will remain in the logical and physical schema.

To delete an existing extension column

1. Connect to your local development database with Siebel Tools.
2. In the Object Explorer, expand the Table object type.
3. In the Object List Editor, select the table containing the extension column to be deleted.
4. In the Object Explorer, select Column.
5. In the Object List Editor, select the extension column to be deleted.

6 Select Edit > Delete from the menu to delete the extension column.

NOTE: Siebel Tools does not cascade the deletion of an extension column. You should delete or inactivate the attribute mapping after deleting the extension column. You can delete mappings by navigating to the Attribute Mapping object in the Object List Editor and select Edit > Delete Record.

To delete an existing extension table

1 Connect to your local development database with Siebel Tools.

2 In the Object Explorer, select the Table object type.

3 In the Object List Editor, select the extension table to delete.

4 Select Edit > Delete from the menu to delete the extension table.

Creating Custom Indexes

You can create custom indexes to enhance the performance of your implementation. You cannot modify standard indexes or delete them from the schema.

When implementing custom indexes, consider the following:

- Typically, new tables require new indexes. Use caution when creating indexes. Custom indexes can cause the standard indexes to not be used and can adversely impact performance.
- If at some point you no longer require a custom index that you have created, do not delete it from the Siebel repository. Instead, inactivate it by selecting the object's Inactive property check box in the Object List Editor.
- Any custom indexes should be thoroughly tested in a test environment before being introduced into production.
- Do not define indexes with names longer than 18 characters in the DB2 environments.

To create a custom index

1 Connect to your local development database with Siebel Tools.

2 In the Object Explorer, expand the Table object type.

3 In the Object List Editor, select the table to which you want to add an index.

4 In the Object Explorer, expand the Index object type.

5 Select Edit > New Record to add a custom index.

For more information on index properties, see *Object Types Reference*.

Do not use index names that are reserved words on your server or client database.

When you add custom indexes to tables, Siebel Tools appends an _X to the index name.

6 In the Object Explorer, select the Index Column object type.

7 Select Edit > New Record to specify each column to add to the index.

Applying Database Extensions to the Local Database

After you have extended the logical schema, by adding columns or tables for example, you must apply those changes to the local database.

NOTE: Siebel eBusiness Applications version 7 do not support customized database triggers. If you have created customized triggers on your Siebel base tables, you must disable them before updating the logical database schema. You will then need to recreate the triggers after the update is finished.

This task is part of the "Select Edit > New Record to specify each column to add to the index." on page 124.

To update your local environment

1 In the Object Explorer, select the table from which you want to apply changes to the database.

2 Click Apply in the Object List Editor.

A dialog box appears, alerting you that you are about to connect to a local database and asking if you want to continue.

NOTE: The Apply button is disabled for tables of type External. For more information about external tables, see *Overview: Siebel eBusiness Application Integration Volume I*.

3 Click OK.

The Apply Schema dialog box appears.

4 Fill in the fields as shown in the following table, and then click Apply.

NOTE: When applying changes, the Privileged User Id must be the tableowner name. For example, suppose your login Id is JSMITH and password is DB2. When you apply schema changes to the local database, your Privileged User Id/Password would be SIEBEL/DB2.

Field	Description
Tables	Select one of the following options from the drop-down menu: ■ **All.** Update the database to reflect all changes made to the dictionary. This option forces each database object to be compared with the data dictionary, and updated if required. ■ **Current Query.** Update the database to reflect modifications made to the tables in the current query only. ■ **Current Row.** Update the database to reflect modifications made to the table in the current row only.
Table space	Leave blank.

Field	Description
16K table space	Leave blank.
32K table space	Leave blank.
Index space	Leave blank.
Table groupings file	Optional. This file is provided by the DBA and is specific to your database.
Privileged user id	Enter your database user ID, typically SIEBEL. The table owner is read from `tools.cfg`.
Privileged user password	Enter your database user password, for example SADMIN. The password is case sensitive and should be all in uppercase. For example, suppose the mobile user JSMITH initialized the local database, he used DB2 as the password. Then when he applies schema changes to the local database, he should use SIEBEL/DB2 as the Privileged User Id/Password pair. When the database initialization for a mobile client is performed, the table owner changes from SIEBEL to the mobile user's password. In this case, use the mobile user's password in the password field.
ODBC data source	Verify that the ODBC connection specified in the ODBC Data Source text box is correct for your environment. You cannot apply schema changes to any database other than the one you are currently connected to (for example, by specifying the ODBC name of a different database).

5 To activate extensions to EIM tables, select the appropriate tables, and then click Activate.

Your extension tables and columns are now available to use in your configuration.

Preparing Server Database for Applying Schema Changes

After you have tested your extensions in the local environment, complete the following actions before applying the changes to the server database:

- Have all mobile users to synchronize.
- Make sure all connected clients are disconnected from the database server.
- Once all mobile user transactions have been merged and routed, stop all Siebel Servers.
- Perform a full backup of the database.

Applying Data Model Changes to the Server Database

After applying data model changes to your local database, testing them, and then checking the projects back in to the server database, you need to apply the changes to the physical server database. Until you do so, only the logical database schema of the server database has been updated.

You can apply data model changes using Siebel Tools or using the Database Configuration Utility.

To apply and activate the changes to the server database Using Siebel Tools

1. Connect to the server.
2. In the Object Explorer, select the table from which you want to apply changes to the database.
3. Click Apply in the Object List Editor.

   A dialog box appears, alerting you that you are about to connect to a local database and asking if you want to continue.
4. Click OK.

   The Apply Schema dialog box appears.

5 Fill in the fields as shown in the following table, and then click Apply.

Field	Description
Tables	Select one of the following options from the drop-down menu: ■ **All.** Update the database to reflect all changes made to the dictionary. This option forces each database object to be compared with the data dictionary, and updated if required. ■ **Current Query.** Update the database to reflect modifications made to the tables in the current query only. ■ **Current Row.** Update the database to reflect modifications made to the table in the current row only.
Table space	Leave blank.
16K table space	Leave blank.
32K table space	Leave blank.
Index space	Leave blank.
Table groupings file	Optional. This file is provided by the DBA and is specific to your database.
Privileged user id	Enter your database user ID, for example SADMIN. The table owner is read from `tools.cfg`.
Privileged user password	Enter your database user password, for example SADMIN. When the database initialization for a mobile client is performed, the table owner changes from SIEBEL to the mobile user's password. In this case, use the mobile user's password in the password field.
ODBC data source	Verify that the ODBC connection specified in the ODBC Data Source text box is correct for your environment. You cannot apply schema changes to any database other than the one you are currently connected to (for example, by specifying the ODBC name of a different database).

NOTE: If you receive an error message and cannot apply changes to the server database, you must use the Database Server Configuration Utility. See "Propagating Changes to Other Local Databases" on page 128.

6 Restart the Siebel Server.

Your extension tables and columns are now available to use in your configuration.

7 Click Activate in the Object List Editor. This increases the custom database schema version and thus prepares for the mobile client upgrade.

Once this process has been completed, any extensions included in the tables you selected will now physically exist on your server database.

Propagating Changes to Other Local Databases

After extensions have been checked into your server database and applied to the physical database you need to propagate the schema changes to other developers (mobile users).

To propagate schema changes to mobile users

1. Have all mobile users perform a full synchronization.
2. Activate the extensions. This procedure differs depending on whether or not you are using Siebel Anywhere.

   Using Siebel Anywhere. Perform the following steps:

   a. Create an Upgrade Kit on your Server database that includes the Siebel Database Schema as the upgrade kit component.

      For more information, see *Siebel Anywhere Administration Guide* for information on creating upgrade kits.

   b. Click Activate on the Upgrade Kits View to make the upgrade kit available.

   Without Siebel Anywhere. Perform the following steps:

   c. Log on to Siebel Tools while connected to your server database.

   d. Click Activate in the Table List view. Executing this process will increase the database schema version number, and therefore require a schema upgrade for mobile users.

3. Run gennewdb to regenerate the template local database.

   For information on running gennewdb, see *Siebel Remote and Replication Manager Administration Guide*.

4. Reextract mobile clients. Mobile clients will need to reinitialize their local databases with the extracted data. This procedure differs depending on whether or not you are using Siebel Anywhere.

   Using Siebel Anywhere. In the Upgrade Configurations View, click Distribute. This action will make the new custom schema version available for a schema upgrade. The Required flag is set manually. See *Siebel Anywhere Administration Guide* for detailed information on Siebel Anywhere Configurations screen.

   Without Siebel Anywhere. Manually reextract and reinitialize all mobile user databases.

5 Configuring EIM Interfaces

Topics in This Chapter

About Interface Tables

Interface tables are intermediate database tables that act as a staging area between the Siebel application database and other databases. Interface tables are typically populated by a DBA using third-party tools, such as SQL Loader. A Siebel Administrator uses Siebel Enterprise Integration Manger (EIM) to perform bulk imports, exports, updates, and deletes.

Interface tables names begin with the prefix EIM_ and they have the Type property set to *Interface*.

To use EIM to populate custom extension tables and extension columns, use the EIM Table Mapping Wizard to create mappings between the new columns and EIM interface tables. For more information, see "EIM Table Mapping Wizard" on page 136.

For detailed information about interface tables and EIM, see *Siebel Enterprise Integration Manager Administration Guide*.

About EIM Object Types

The mappings between columns and tables and EIM interface tables is implemented using a hierarchy of EIM object types, shown in Figure 18.

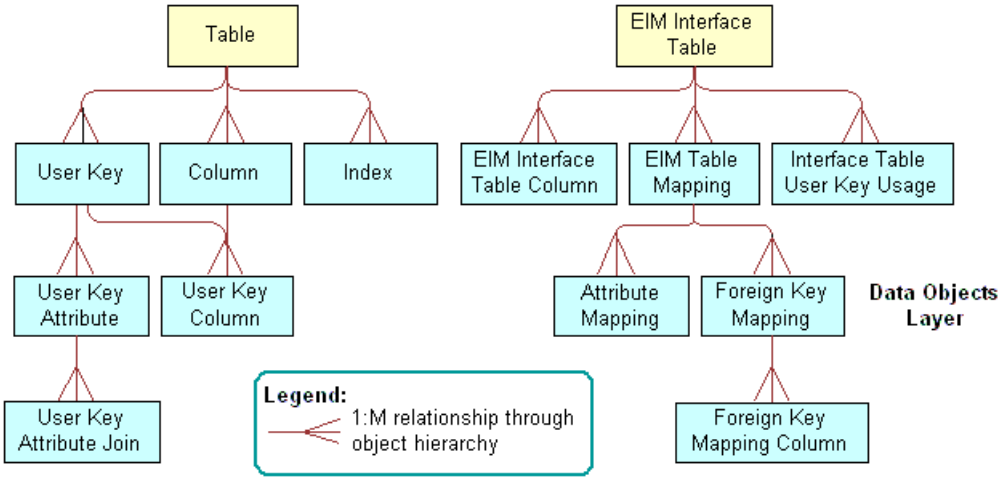


Figure 18. EIM Architecture

EIM Interface Table Object Type

The EIM Interface Table object type is an alternative representation of the Table object type. Each interface table has a table object definition (with a value of Interface in the Type property) and an EIM interface table object definition, as shown in Figure 19.

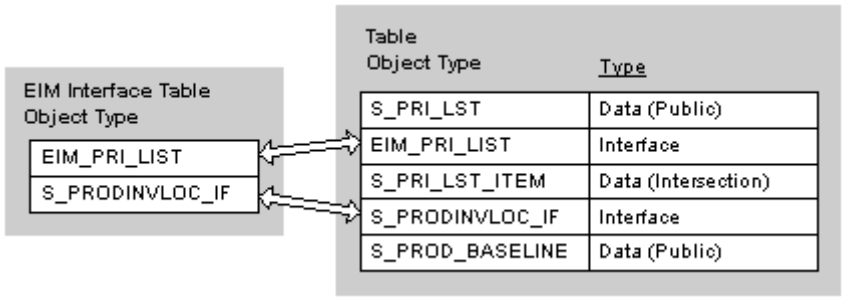


Figure 19. Relationship Between EIM Interface Table and Table

The EIM Interface Table object type has the properties of the Table object type, plus several additional properties that are specific to interface tables. EIM Interface Table has the following child object types: EIM Interface Table Column, EIM Table Mapping, and Interface Table User Key Usage.

EIM Interface Table Column Object Type

The EIM Interface Table Column object type is an alternative representation of the Column object type, for columns that are child object definitions of interface tables. For a given interface table, the same list of columns appears as column children of the table object definition and as EIM interface table column children of the corresponding EIM interface table object definition. This is shown in Figure 20.

The EIM Interface Table Column object type contains all the properties of the Column object type, in addition to some that are specific to EIM.

NOTE: The Price List interface table EIM_PRI_LST is used in this and subsequent examples in the section.

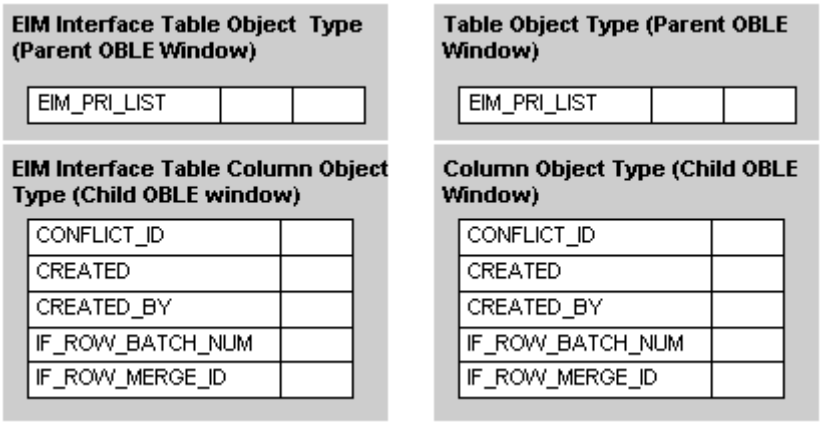


Figure 20. Relationship Between EIM Interface Table Column and Column

Interface Table User Key Usage Object Type

The Interface Table User Key Usage object type provides support for alternative user keys for base tables. An interface table user key usage object definition defines the use of a nontraditional user key for a given base table in a specific interface table.

NOTE: Do not modifying user keys in standard Siebel tables or EIM base tables.

EIM Table Mapping Object Type

Identifies a data table that is updated by the parent EIM interface table object definition. One interface table may update one or more data tables, and each data table to be updated requires an EIM Table Mapping child object definition of the EIM Interface Table object. Each EIM Table Mapping object definition identifies the name of the destination table (data table to update) in its Destination Table property. This is illustrated in Figure 21.

EIM Table Mapping has two child object types: Attribute Mapping and Foreign Key Mapping.

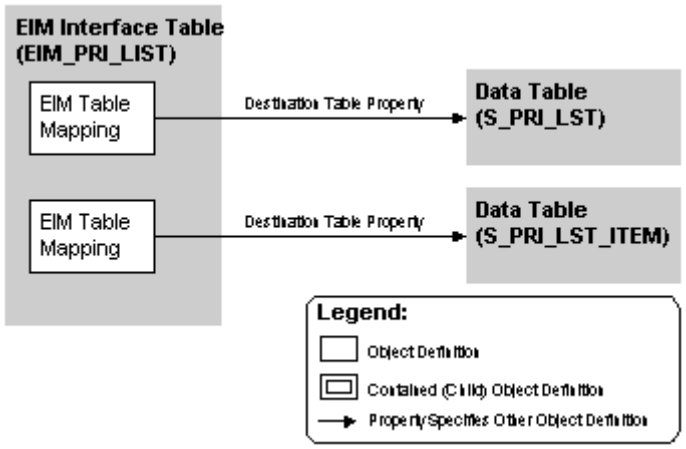


Figure 21. EIM Table Mapping Configuration

Attribute Mapping Object Type

Identifies an attribute (data) column to update in the destination (base) table specified in the parent EIM table mapping. Each Attribute Mapping object definition identifies the column in the interface table that supplies the data (in the Interface Table Data Column property). It also identifies the column in the destination table that receives the data (in the Base Table Attribute Column property).

You can configure the Attribute Mapping object type. You should add a corresponding Attribute Mapping object definition when you add an extension column to a table, if the extension table is to be populated by an interface table.

The Attribute Mapping object type is shown in Figure 22.

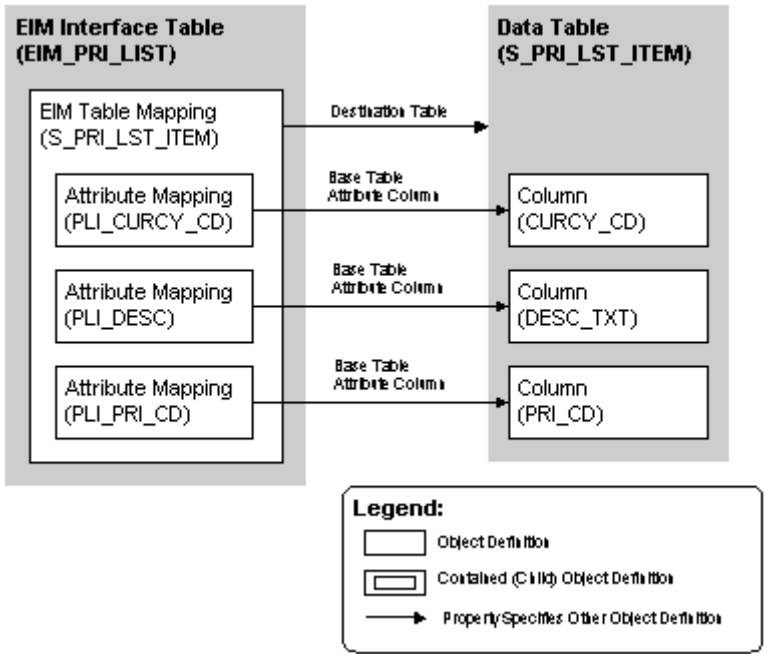


Figure 22. Attribute Mapping Configuration

Foreign Key Mapping Object Type

Each Foreign Key Mapping object definition identifies a foreign key column in the destination table that is to be populated from the interface table. Because foreign key values are stored as numeric row ID values in data tables, to populate one from an interface table it is necessary to map from the interface column to a combination of user key columns in the destination table, rather than directly to the foreign key column.

A foreign key mapping is not a one-to-one column mapping from interface table to destination table, as occurs with non-foreign key columns. The numeric foreign key does not even exist in the interface table, so it cannot be mapped. Instead, a combination of attribute columns in the destination table of the foreign key is used to access the desired row, and the foreign key value can be obtained from that row. These relationships are illustrated in Figure 23 on page 134.

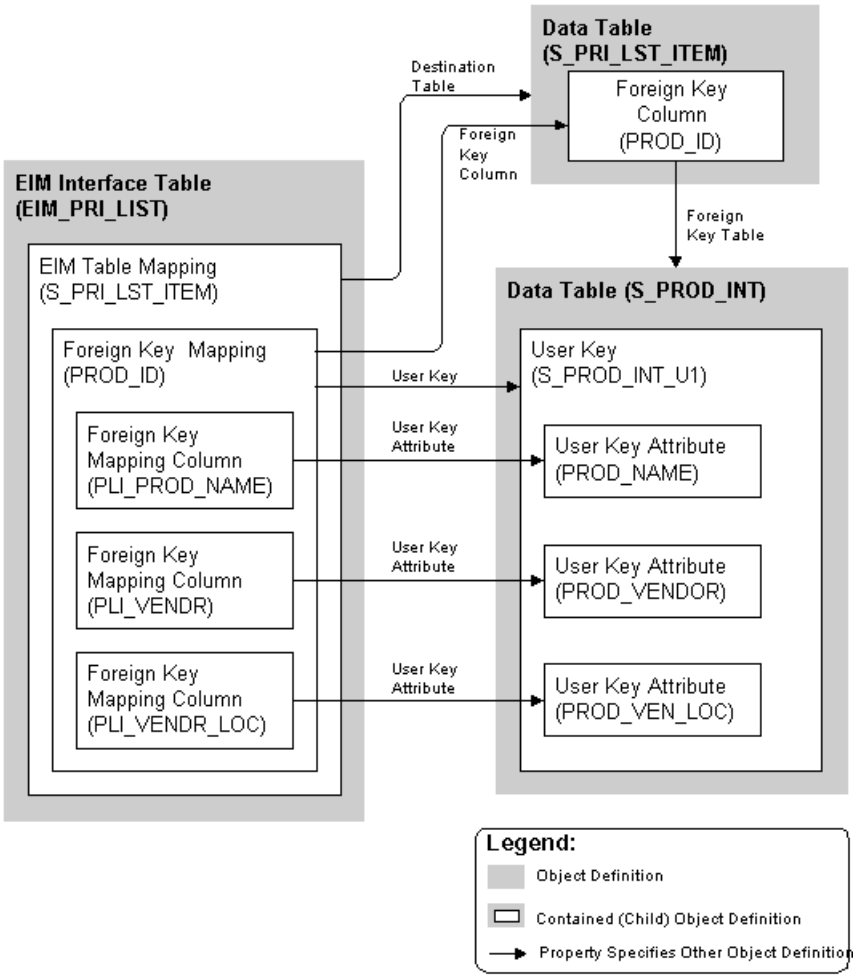


Figure 23. Foreign Key Mapping Configuration

Foreign Key Mapping Column Object Type

Each Foreign Key Mapping Column object definition identifies a piece of the user key columns; that is, one of the attribute columns used to locate rows in the table the foreign key points to. The user key columns, taken together, uniquely identify rows in that table. The Foreign Key Mapping Column object definitions identify these user key columns to the interface table, so that foreign key values can be derived when import or export takes place.

User Key Object Type

User Key is a child object type of Table. Each user key object definition provides a set of attribute columns and related information that specifies how the table's rows can be accessed in a particular EIM scenario. User Key has two child object types: User Key Column and User Key Attribute.

User Key Column Object Type

User key columns can be either attributes or foreign keys. In most cases these are the columns in the user key index (usually the index with a suffix of _U1), with the exception of the CONFLICT_ID column.

User Key Attribute Object Type

Each user key attribute object definition in the parent user key specifies one in the set of attribute columns that collectively identify rows in the grandparent table. The column name is specified in the Name property of the User Key Attribute object definition. User Key Attribute has one child object type, which is User Key Attribute Join.

User Key Attribute Join Object Type

Each User Key Attribute object definition has one or more User Key Attribute Join child object definitions. The user key attribute join specifies a join operation that can be used to convert a user key attribute that is itself a foreign key to another table into attribute column values in that table. For example, the S_PROD_INT (products) table has a user key consisting of three attributes: PROD_NAME, PROD_VENDOR, and PROD_VEN_LOC. The PROD_NAME (product name) attribute column is directly obtained from the S_PROD_INT table, so no join is required. However, the PROD_VENDOR and PROD_VEN_LOC columns occur in the S_ORG_EXT (accounts) table, and must be obtained using a join on VENDR_OU_ID, a foreign key from S_PROD_INT to S_ORG_EXT.

Labeling Data Loaded in EIM As No Match Row Id Instead of NULL

When you are loading data through EIM and a primary child column has no match, it is labeled as NULL, whereas loading it through the user interface would produce No Match Row Id.

To fix the problem you need to open the record set in the client user interface and manually step through each record created by EIM—each instance of a NULL value for PR_TERR_ID will be replaced with No Match Row Id.

For more information, see "Using the Check No Match Property with a Primary Join" on page 210.

EIM Table Mapping Wizard

You use the EIM Table Mapping Wizard to map custom columns and tables to EIM interface tables. Before adding and modifying attribute mappings consider the following:

- You can map a single column in an interface table to multiple base tables or extension tables. However, do not map multiple interface table columns to a single column in a target table because it can create ambiguity for EIM.
- The EIM Table Mapping Wizard is only available to tables of type Data (Public), Data (Intersection), Extension, and Extension (Siebel).
- You cannot use the EIM Table Mapping Wizard with non-standalone customer tables or foreign keys to custom tables. This is because you cannot add user key attributes and the EIM Table Mapping Wizard does not map foreign key columns if they point to a parent table column that does not have user key attributes.
- To invoke the EIM Table Mapping Wizard for Siebel base tables that do not have the foreign key as part of the user key, you need to create a temporary column with the following properties:

Field	Value
Inactive	Y
User Key Sequence	<> NULL (for example, set it to 0)
FK (Foreign Key)	Set (FK) Foreign Key table as itself

  By creating this temporary column, when you launch the EIM Mapping Wizard, it will list standard EIM interface tables that are already mapped to this table as the target or destination tables. The wizard will also list EIM tables that are mapped to tables to which this table has a foreign key. However, the foreign key must be part of the "Traditional U1 Index" user key of this table. After the EIM Mapping Wizard finishes, you should delete this temporary column.

- Review the restrictions listed in Table 17.

Table 17. Restrictions on Adding and Modifying Mappings

From	To	Conditions
Interface table column	Base column	Supported if there are existing mappings from this interface table into this data table.
Interface table extension column	Base column	Supported if there are no other mappings to this base column. Use with caution.
Interface table column	Extension table column	Supported if there are existing mappings from this interface table into the extension table's base table.
Interface table extension column	Extension table column	Supported if there are existing mappings from this interface table into the extension table's base table.

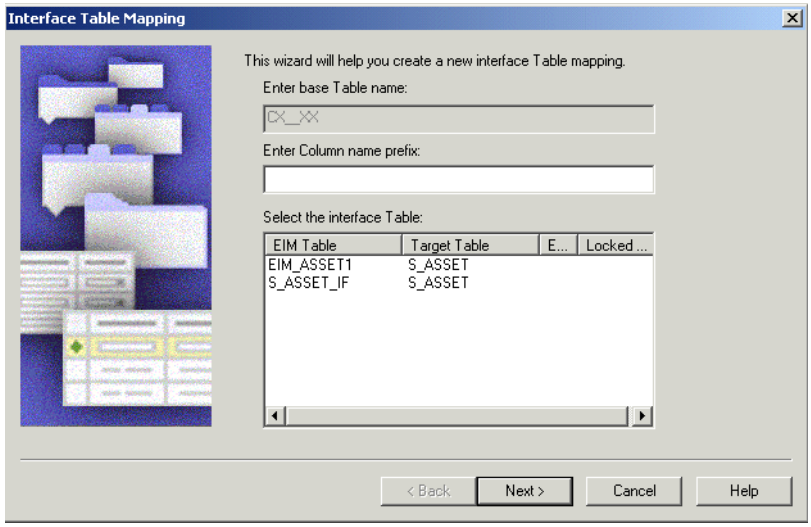
- You can deactivate mappings if they are no longer necessary. To deactivate a mapping, navigate to the Attribute Mapping Object definition in the Object List Editor and place a check mark in the Inactive property. You should not delete any mappings.
- No validation is performed against interface table or column definitions. LOV validation is performed against the LOVs defined for the base columns to which they are mapped.

To map a new table to an EIM interface table using the EIM Table Mapping Wizard

1. Lock the project.
2. Select Table object type in the Object Explorer.
3. In the Object List Editor, select a table that you want to map to an EIM Table.

   This table will be the primary table into which data from the existing Interface table will be imported.
4. Right-click and choose EIM Mapping Table from the menu.

   The Interface Mapping dialog box is displayed with the Base Table name field populated with the selection you made in the Object List Editor.



5. In the *Edit the Column name prefix* field, enter a distinguishing prefix.

   If a prefix does not already exist for the selected EIM table, the new prefix will be added to specified EIM Interface Table Columns related to the target table. If a prefix already exists, the existing prefix will be used.

6 In the *Select an interface table* field, click the drop-down arrow and then choose a value from the list.

The list for selecting the EIM Interface Table is constrained to show interface tables that are mapped to tables to which your new custom table has a foreign key relationship.

The list of candidate interface tables is sorted by EIM table name. Interface tables with EXIST=Y means that these EIM tables already have the base table mapped. If you extend existing Siebel tables, consider these tables the ideal candidates for EIM mapping.

7 Click Next on the Interface Table Mapping dialog box.

The Summary dialog box appears with a summary of the choices you have made.

8 Click Finish on the Summary dialog box to accept the choices you made and generate the EIM Interface Table object.

Based on this information, the wizard creates new EIM table mapping objects and adds several child objects to an existing EIM interface table object, as shown in Figure 24.

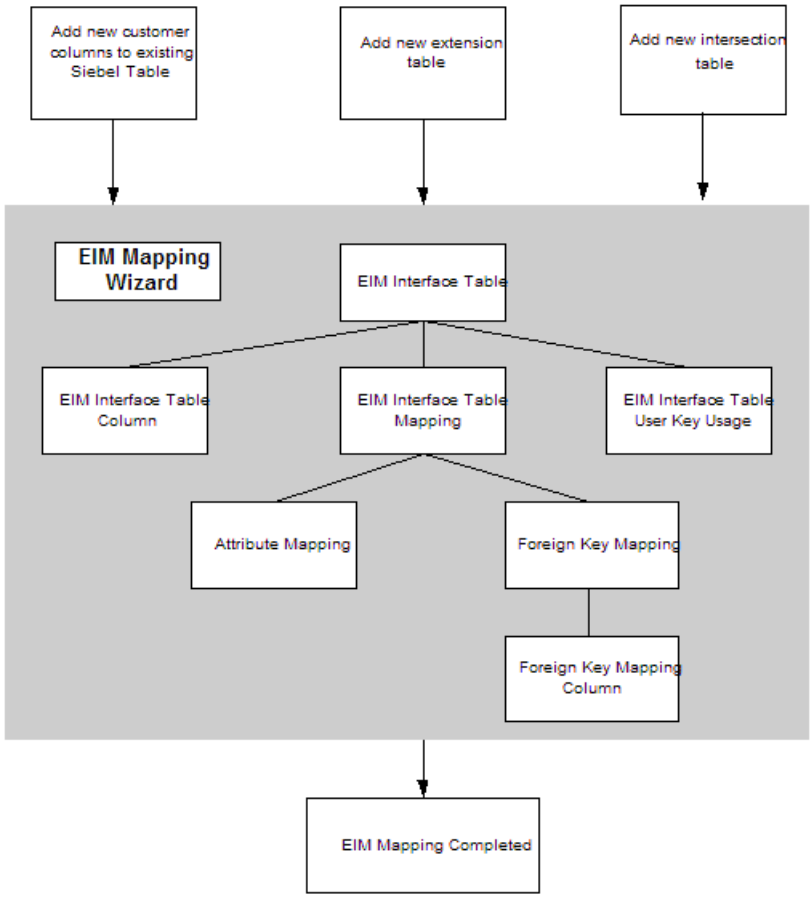


Figure 24. EIM Mapping

For detailed specifications about EIM objects created, see "EIM Object Specifications" on page 139.

EIM Object Specifications

The topics in this section detail objects specifications that EIM uses.

For detailed information about EIM, see *Siebel Enterprise Integration Manager Administration Guide*.

EIM Interface Table

EIM interface table object specifications are summarized in Table 18.

Table 18. EIM Interface Table Object Specifications

Specification	Value
Target Table	Selected by the developer
EIM Delete Proc Column	T_DELETED_ROW_ID
EIM Export Proc Column	T_EXPORTED_ROW_ID
EIM Merge Proc Column	T_MERGED_ROW_ID

EIM Interface Table Columns

EIM Interface Table Column specifications are covered in Table 19.

Table 19. System Columns on the EIM Interface Table

Name	Physical Type	Length	Type	EIM Processing Column
CONFLICT_ID	Varchar	15	System	FALSE
CREATED	Date Time	7	System	FALSE
CREATED_BY	Varchar	15	System	FALSE
IF_ROW_BATCH_NUM	Number	22	System	FALSE
IF_ROW_MERGE_ID	Varchar	15	System	FALSE
IF_ROW_STAT	Varchar	30	System	FALSE
IF_ROW_STAT_NUM	Number	22	System	FALSE

Table 19. System Columns on the EIM Interface Table

Name	Physical Type	Length	Type	EIM Processing Column
LAST_UPD	Date Time	7	System	FALSE
LAST_UPD_BY	Varchar	15	System	FALSE
MODIFICATION_NUM	Number	22	System	FALSE
ROW_ID	Varchar	15	System	FALSE

Generic EIM Interface Table Columns for EIM Processing

For each EIM Table Interface, the columns shown in Table 20 are created to facilitate processing. You cannot change the values of these columns.

Table 20. Generic EIM Interface Table Columns for EIM Processing

Name	Physical Type	Length	Type	User Name	EIM Processing Column
T_DELETED_ROW_ID	Varchar	15	Data (Private)	Deleted ROW_ID from base table	TRUE
T_EXPORTED_ROW_ID	Varchar	15	Data (Private)	Exported ROW_ID from target table	TRUE
T_MERGED_ROW_ID	Varchar	15	Data (Private)	Merged into ROW_ID from target table	TRUE

EIM Interface Table Columns for Processing a Mapping to a Specified Table

There are five of these columns for each EIM Table Mapping object, with the properties listed in Table 21.

Table 21. EIM Interface Table Columns for Processing a Particular Mapping

Column	Value
Name	Derived from the name of the target table: "T_"+ [EIM Table Mapping Name without the "CX_"]+ "_"+ [Process-specific suffix]
Physical Type	Depends on the process to which the column is related
Length	Depends on the process to which the column is related

Table 21. EIM Interface Table Columns for Processing a Particular Mapping

Column	Value
Type	Depends on the process to which the column is related
User Name	Name of the EIM Table Mapping object for which the column is being created
EIM Processing Column	TRUE

For example, if the target table selected by the user is CX_SEC_LEV, an EIM Table Mapping object is created. The following four column objects are generated, with the corresponding default properties (Table 22):

Table 22. EIM Interface Table Columns for Processing a Mapping to CX_SEC_LEV

Name	Physical Type	Length	Type	User Name	EIM Processing Column
T_SEC_LEV_EXS	Character	1	IFMGR: Exists	CX_SEC_LEV	TRUE
T_SEC_LEV_RID	Varchar	15	IFMGR: ROW_ID	CX_SEC_LEV	TRUE
T_SEC_LEV_STA	Number	22	IFMGR: Status	CX_SEC_LEV	TRUE
T_SEC_LEV_UNQ	Character	1	IFMGR: Unique	CX_SEC_LEV	TRUE

EIM Interface Table Columns for Foreign Key Processing

A column is created for each foreign key on the relevant EIM Table Mapping object (that is, the target table). These columns have the properties listed in Table 23.

Table 23. EIM Interface Table Columns for Foreign Key Processing

Column	Value
Name	Derived from the target table name and the corresponding foreign key column on the target table in the following format: [Target Table name (with the "CX" prefix replaced by "T")]+[Target table foreign key column]
Type	Set to IFMGR: Fkey
Physical Type	Physical type of foreign key column on Target Table (typically Varchar)
Length	Length of foreign key column on Target Table (typically 15)
User Name	[Target Table (or EIM Table Mapping) name]+"."+[Foreign key column name]

For example, if CX_SEC_LEV contains foreign key columns called OPTY_ID and ACCNT_ID to the S_OPTY and S_ORG_EXT tables, respectively, the following EIM Table Columns are generated as shown in Table 24.

Table 24. EIM Interface Table Columns for Processing Foreign Keys in CX_SEC_LEV

Name	Physical Type	Length	Type	User Name
T_SEC_LEV_OPTY_ID	Varchar	15	IFMGR: Fkey	CX_SEC_LEV.OPTY_ID
T_SEC_LEV_ACCNT_ID	Varchar	15	IFMGR: Fkey	CX_SEC_LEV.ACCNT_ID

EIM Interface Table Columns for Foreign Keys

A separate foreign key column will be created for each U1 user key column on the foreign key tables. The columns have the following properties listed in Table 25.

Table 25. EIM Interface Table Columns for Foreign Keys

Column	Value
Name	[First four letters of foreign key table name without the "S_" prefix, trimmed to remove any trailing "_"characters] + "_"+[foreign key column name on target table]
Physical Type	Physical type of the user key column on the target table (typically Varchar)
Length	Length of these columns will correspond to the length of user key columns upon which they are based (typically 15)
Type	Data (Public)

Continuing with the CX_SEC_LEV example the columns would have the properties listed in Table 26.

Table 26. EIM Interface Table Columns for CX_SEC_LEV Foreign Keys

Name	Physical Type	Type
OPTY_BU_ID	Varchar	Data (Public)
OPTY_NAME	Varchar	Data (Public)
OPTY_PR_DEPT_OU_ID	Varchar	Data (Public)
ORG_BU_ID	Varchar	Data (Public)
ORG_NAME	Varchar	Data (Public)
ORG_LOC	Varchar	Data (Public)

NOTE: Depending on the base column type, corresponding EIM columns are generated accordingly.

EIM Interface Table Column for Each Attribute on the Target Table

Attribute columns on the target table are those of type Data (Public) that have a null Foreign Key Table property. These EIM interface table columns will have the properties listed in Table 27.

Table 27. EIM Interface Table Columns for Each Attribute on the Target Table

Column	Value
Name	[Prefix entered—for example, CON or ACCNT]+"_"+[name of the corresponding column in the target table]
Physical Type	Data (Public)
Length	Length of corresponding column in the target table
User Name	Name of corresponding column in the target table

For example, if you enter a prefix of SECL and have the following attribute columns in CX_SEC_LEV: NAME (Varchar 100), DESC_TEXT (Varchar 250), and AUTO_UPDATE (Char 1), the following EIM interface table columns are generated as shown in Table 28.

Table 28. EIM Interface Table Columns for Each Attribute on CX_SEC_LEV

Name	Physical Type	Length	Type	User Name
SECL_NAME	Varchar	100	Data (Public)	Security Level Name

Table 28. EIM Interface Table Columns for Each Attribute on CX_SEC_LEV

Name	Physical Type	Length	Type	User Name
SECL_DESC_TEXT	Varchar	250	Data (Public)	Security Level Description
SECL_AUTO_UPDATE	Char	1	Data (Public)	Auto Update Flag

EIM Table Mapping Objects Based on the Target Table

The name and destination columns will be set to the name of the target table. The processing column properties will correspond to those that have been automatically generated. For examples, see Table 29.

Table 29. EIM Table Mapping Objects Based on the Target Table

Column	Value
Name	CX_SEC_LEV
Destination Table	CS_SEC_LEV
EIM Exists Proc Column	T_SEC_LEV_EXS
EIM Row Id Proc Column	T_SEC_LEV_RID
EIM Status Proc Column	T_SEC_LEV_STA
EIM Unique Proc Column	T_SEC_LEV_UNQ

Attribute Mapping objects for Each EIM Interface Column Generated

Attribute Mapping objects should have the property values listed in Table 30 for each EIM interface column generated:

Table 30. Attribute Mapping Objects for Each EIM Interface Column

Object	Value
Name	Attribute column on target table
Interface Table Data Column	Name of corresponding EIM interface table column generated (Table 28 on page 144)
Base Table Attribute Column	Name of attribute column on target table

Foreign Key Mapping for Each Foreign Key Column on the Target Table

A separate Foreign Key Mapping object is created for each foreign key mapping column on the target table. The properties listed in Table 31 are set for each.

Table 31. Foreign Key Mapping for Each Foreign Key Column on a Target Table

Object	Value
Name	Name of the user key column
Foreign Key Column	Name of the user key column
User Key	Name of the U1 user key of the foreign key table
EIM Foreign Key Proc Column	Corresponding EIM interface table column for foreign key processing: "T_"+[Target table name without "CX_" prefix] +"_"+[user key column name]

Continuing with the CX_SEC_LEV example, Table 32 shows the foreign key mapping.

Table 32. Foreign Key Mapping for Each Foreign Key Column on CX_SEC_LEV

Name	Foreign Key Column	User Key	EIM Foreign Key Proc Column
OPTY_ID	OPTY_ID	S_OPTY_U1	T_SEC_LEV_OPTY_ID
ACCNT_ID	ACCNT_ID	S_ORG_EXT_U1	T_SEC_LEV_ACCNT_ID

Foreign Key Mapping Columns for each Foreign Key Mapping object

A separate Foreign Key Mapping Column object is created for each user key column in the user key specified for the parent Foreign Key Mapping object (Table 31 on page 146). The following properties are set for each foreign key mapping column (Table 33):

Table 33. Foreign Key Mapping Columns for Each Foreign Key Mapping Object

Column	Value
Name	Name of the Foreign Key Mapping Column.

Table 33. Foreign Key Mapping Columns for Each Foreign Key Mapping Object

Column	Value
Interface Data Column	EIM interface table column to which the user key column on the target table should be mapped. This EIM interface table column is generated according to the specifications in Table 25 on page 143.
User Key Attribute	Name of the corresponding user key column that belongs to the user key specified in Table 31 on page 146.

Continuing with the CX_SEC_LEV example, Table 34 shows the foreign key mapping.

Table 34. Foreign Key Mapping Columns for CX_SEC_LEV

Name	Interface Data Column	User Key Attribute
OPTY_BU_ID	OPTY_BU_ID	BU_ID
OPTY_NAME	OPTY_NAME	NAME
OPTY_PR_DEPT_OU_ID	OPTY_PR_DEPT_OU_ID	PR_DEPT_OU_ID
ORG_BU_ID	ORG_BU_ID	BU_ID
ORG_NAME	ORG_NAME	NAME
ORG_LOC	ORG_LOC	LOC

6 Configuring Docking Rules

Topics in This Chapter

About Dock Objects

Siebel applications can selectively replicate server data into the local database, and thus support mobile computing by allowing field personnel to share information across the enterprise, including both mobile and connected users. This is accomplished using Siebel Remote.

Dock objects are the foundation of Siebel Remote. Dock objects are collections of related tables. Each Dock Object object in Siebel Tools has one primary table and several other related tables as Dock Object Table child objects.

Different data is replicated to different local databases, depending on each local database owner's employee identity, position, organization, and visibility to data from different dock objects, and on the relationship between the dock objects. These rules are known collectively as routing rules or Dock Object Visibility Rule child objects.

When the data is updated on the server, the local database is synchronized when the mobile user connects to the Siebel Server and performs the synchronization, but only the data that should be replicated into the local database is synchronized. During the synchronization, any updates in the local database are also uploaded to the server.

For more information on routing rules and synchronization, see *Siebel Remote and Replication Manager Administration Guide*.

Dock Object Types

There are three types of dock objects:

- **Private.** Private dock objects are used exclusively for routing of non-configurable data. This setting makes sure that the rows in these dock objects are never routed to any mobile clients. All records from tables that are part of a private dock object are uploaded to the server during synchronization. None of these records are downloaded to remote users.

- **Enterprise.** Enterprise dock objects involves a distribution of records without restriction. All records from tables that are part of an Enterprise dock object are uploaded to the server during synchronization. Most of these tables should only be updated by an administrator and are typically downloaded but not uploaded by mobile users. To minimize synchronization time, Enterprise dock objects should only be used with tables that contain small volumes of data or are semi-static in nature (that is, the contents change infrequently).
- **Limited.** Limited dock objects contain numerous individual rules for determining the records that should be downloaded to particular users; these users should only get those records with which they have some direct or indirect involvement.

  There are nine types of visibility rules used for Limited dock objects:

  - **Employee.** Evaluates record according to whether it has a foreign key to the mobile user's Employee record.
  - **Employee Manager.** Evaluates record according to whether it has a foreign key to the Employee record of someone who directly reports to the mobile user.
  - **Position.** Evaluates record according to whether it has a foreign key to the mobile user's primary Position record.
  - **Position Manager.** Evaluates record according to whether it has a foreign key to the Position record of someone who directly reports to the mobile user.
  - **Organization.** Evaluates record according to whether it is associated with the same business unit as the mobile user.
  - **Check Dock Object.** Evaluates record according to whether it is related to another record that the user receives.
  - **Calendar.** Applies only to calendar appointment records. Evaluates record according to whether the mobile user has access to the calendar of the record's owner.
  - **Category.** Evaluates record according to whether it is in a category visible to the user.
  - **SQL Rule.** Used to handle special exceptions through custom SQL.

For more information, see "Dock Object Visibility Rules" on page 152.

Dock Object Table

The Dock Object Table object type is a child object type of Dock Object, and is used to specify the tables whose records are actually transferred in conjunction with the Dock Object. The Opportunity dock object and its child dock object tables are shown in Figure 25.

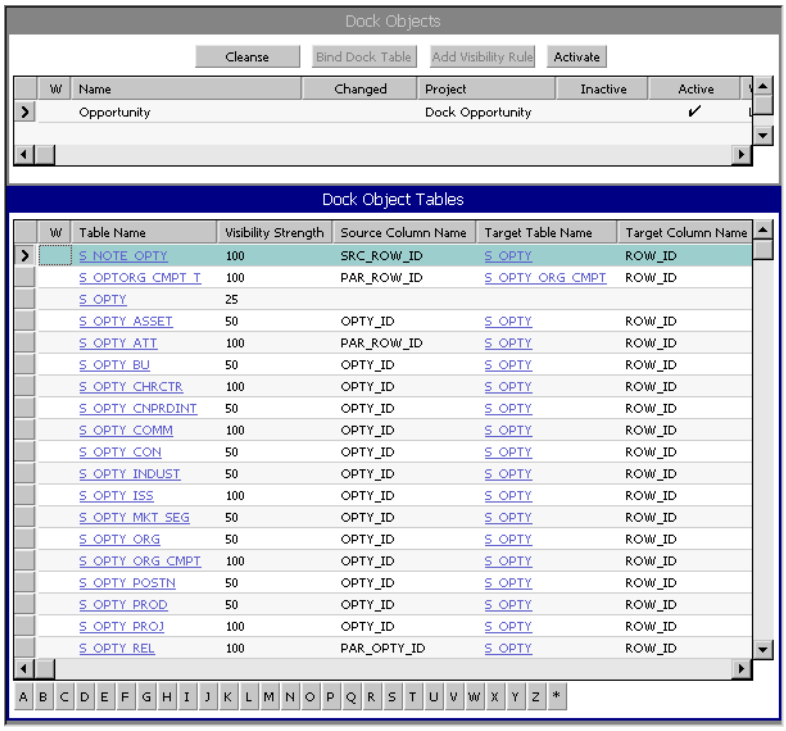


Figure 25. Dock Object and Dock Object Tables

All of the tables identified in dock object tables for a given dock object are related, through foreign keys in the data model, to one *driving table* (also represented by a Dock Object Table object definition). The driving table is identified in the Primary Table property in the Dock Object object type.

For example, the Opportunity dock object shown in Figure 25 on page 151 is based on the primary table S_OPTY, but it also includes other dock object tables such as S_NOTE_OPTY (notes for the opportunity) and S_OPTY_REL (relationships between opportunities). Also included are the extension tables for S_OPTY.

A dock object is therefore a set of logical records (opportunities in this case), where each such logical record is itself a collection of one or more physical database records spread across multiple tables.

Dock Object Visibility Rules

To determine which records in a dock object to download to each mobile user, the Siebel application evaluates the dock object visibility rules for that dock object. Dock Object Visibility Rule is a child object type of Dock Object, as illustrated for Opportunity in Figure 26.

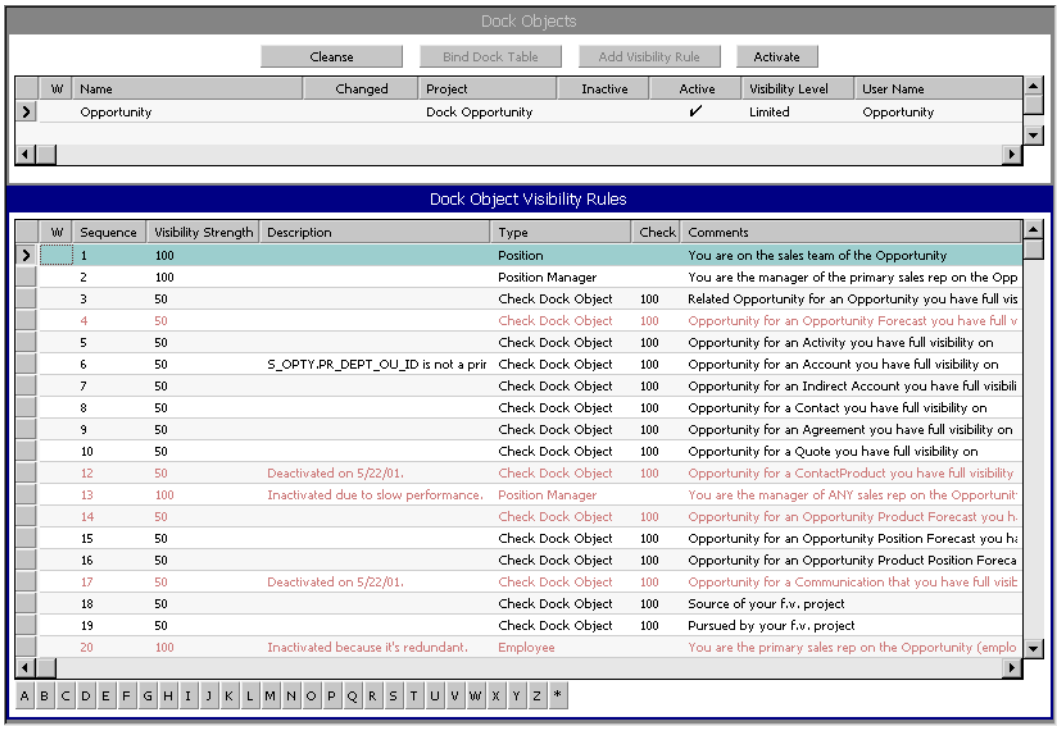


Figure 26. Dock Object and Dock Object Visibility Rules

Each visibility rule has a Comment property that explains specifically what the rule checks. For example, the dock object visibility rules on the Opportunity dock object include the following: "You are on the sales team of the Opportunity," "You are the manager of the primary sales rep on the Opportunity's sales team," and so on. In addition, each dock object visibility rule has a Visibility Strength property and a Sequence property.

Siebel applications determine which database records to propagate to each mobile user (for dock objects that have limited visibility) by evaluating the *visibility strength* of the user for a dock object, and comparing this with the visibility strengths of the tables it contains.

The user's visibility strength for a dock object is determined from the child dock object visibility rules. For each dock object record, and for each mobile user, the Siebel application sequentially evaluates the rules in order of descending visibility strength and ascending sequence until one of them "passes" (that is, evaluates to TRUE). As soon as one of the rules passes, the Siebel application stops this evaluation process, and gives the current dock object record to the current mobile user.

When a dock object visibility rule passes, the mobile user gets the parent dock object record with a visibility strength value obtained from the corresponding property in the dock object visibility rule that caused him or her to get the record. If none of the dock object visibility rules passes for a given dock object record and a given mobile user, then that user will not receive that particular record.

For example, consider two different dock object visibility rules in the Opportunity dock object:

- The first visibility rule ("You are on the sales team of the Opportunity") has a visibility strength of 100.
- The sixth rule ("Opportunity for an Account you have full visibility on") has a visibility strength of 50.

If users are on the sales team for a particular opportunity, they get that opportunity record with a visibility strength of 100. However, if they are not on the sales team for that opportunity—and if the next four visibility rules also fail—they still get the opportunity record with a visibility strength of 50 if they have full visibility to the account for that opportunity.

Visibility strength values are integers between 0 and 100. A visibility strength of 100 denotes full visibility, while a visibility strength of 0 denotes no visibility. Any value between 1 and 100 (typically 25 or 50) implies partial visibility.

NOTE: The integer range for a visibility strength value is actually 0–254, but a value of 100 is, by convention, considered to mean full visibility. If your configuration does not require the use of values higher than 100, use values in the 0–100 range rather than 0–254.

The user's visibility strength (obtained from the successful dock object visibility rule) is compared with each dock object table's visibility strength, as specified in its Visibility Strength property. For users to receive the records from a particular dock object table, their visibility strength must be greater than or equal to the visibility strength specified for that table.

For example, suppose that a particular mobile user receives a particular logical record from the Opportunity dock object with a visibility strength of 50. The Siebel application will then propagate to the user's local database all physical records that are related to the given opportunity on any of the dock object tables that have a visibility strength less than or equal to 50 in the Opportunity dock object.

For more information on routing (visibility) rules and their implementation, see the chapter about Siebel Remote administration in *Siebel Remote and Replication Manager Administration Guide*.

Finding a Dock Object for a Business Component

Dock objects are provided for standard Siebel applications. Review the standard dock objects and associated visibility rules thoroughly to see if they satisfy a desired visibility change.

Some important business components and their associated dock objects are shown in Table 35.

Table 35. Important Business Components and Their Dock Objects

Business Component	Dock Object	Primary Table	Visibility Level
Action	Activity	S_EVT_ACT	Limited
Account	Party	S_PARTY	Limited
Asset Mgmt - Asset	Asset	S_ASSET	Limited
Contact	Party	S_PARTY	Limited
Employee	Party	S_PARTY	Limited
Opportunity	Opportunity	S_OPTY	Limited
Position	Party	S_PARTY	Limited
Internal Product	Product	S_PROD_INT	Limited
Service Request	ServiceRequest	S_SRV_REQ	Limited

NOTE: Although the Employee and Position records are represented by the Party dock object, which is of Limited visibility, the employee and position records themselves have Enterprise visibility, based on SQL Rule type rules within the Party dock object.

To determine the dock object to which a business component belongs

1. In the Object Explorer, select the Business Component object, and then query for the desired business component.
2. Note the Table property of the business component.

   For example, the base table of Opportunity is S_OPTY.
3. In the Object Explorer, select the Flat tab, select the Dock Object object, and then query in the Dock Object Table field for the table.

   If the table belongs to a dock object, that dock object will be listed. For example, the dock object to which the Opportunity business component belongs is Opportunity.

Docking Wizard

The Docking Wizard is used to extend Siebel Remote functionality to support custom database schema changes. You can use the Docking Wizard to do the following:

- Create new dock objects for custom extension tables that are not already in a dock object.
- Create new dock object tables for custom dock objects.

- Create new dock object visibility rules for custom and existing dock objects.

  NOTE: This is not done directly by the user. The appropriate visibility rules will be added to the dock object depending on the visibility type of the dock object and the structure of the tables involved.

  New dock object visibility rules can be added as a result of one of two actions:

  - A table is added to a custom dock object as a dock object table using the Docking Wizard.
  - The Docking Wizard is invoked from a custom extension column that acts as a foreign key to another table.

The Docking Wizard automatically creates or updates Dock Object, Dock Object Table, and Dock Object Visibility Rule objects for custom tables. You can create Public, Private, and Limited dock objects through the Docking Wizard.

The Docking Wizard creates Limited dock object visibility rules of the following types:

- **Employee.** Employee rules replicate data depending on the mobile user's employee identity. To find all candidate rules, find all columns that are foreign keys to S_USER table, except CREATED_BY and LAST_UPD_BY.
- **Employee Manager.** Employee Manager rules replicate data based on which employees report to the mobile user. The algorithm for finding all candidate rules is the same as for Employee rules.
- **Position.** Position rules replicate data based on the position the mobile user holds. To find all candidate position rules, the algorithm finds all columns that are foreign keys to the S_POSTN table.
- **Position Manager.** Position manager rules replicate data based on which positions report to the mobile user position. The algorithm to find all candidate rules is the same as for Position rules.
- **Check Dock Object.** Check Dock Object rules replicate data depending on which piece of data from other dock objects is replicated to the mobile local database. The relationship between data in other dock objects and the current dock object determine which records from the current dock object are replicated.

  The Docking Wizard can only find the candidate Check Dock Object rules based on the "Foreign Key Table Name" property definitions for columns. For each foreign key, there are two candidate Check Dock Object rules, regardless of where the foreign key column resides:

  - Rules that use this dock object as the destination dock object. There are two types of these rules:
    - **Based on foreign keys on the primary table of the current dock object.** The algorithm to find this kind of candidate rule must find in the table of the current dock object all foreign key columns, other than those pointing to S_USER or S_POSTN. For these foreign key columns, the algorithm needs to find the foreign key table to which these foreign key columns refer. The dock object of the foreign key table will become the Check Dock Object object of the newly created Check Dock Object rule in the current dock object.

- **Based on foreign keys on the primary table of other dock objects.** To find this type of candidate rule, the algorithm must find all foreign key columns that refer to the primary table of the current dock object, on any table that is part of a limited dock object. The algorithm will add the appropriate Check Dock Object visibility rules to these limited dock objects, with the current dock object being the Check Dock Object object.

- Rules that use this dock object as the source dock object, that is, Check Dock Object rules. There are two types of these rules:
  - Based on foreign keys on the primary table of the current dock object.
  - Based on foreign keys on the primary table of other dock objects.

  The algorithm for these types of rules is similar to the algorithm for rules that use this dock object as the destination dock object. The main difference involves switching the source table or column and target table or column.

The Docking Wizard process flow is shown in Figure 27.

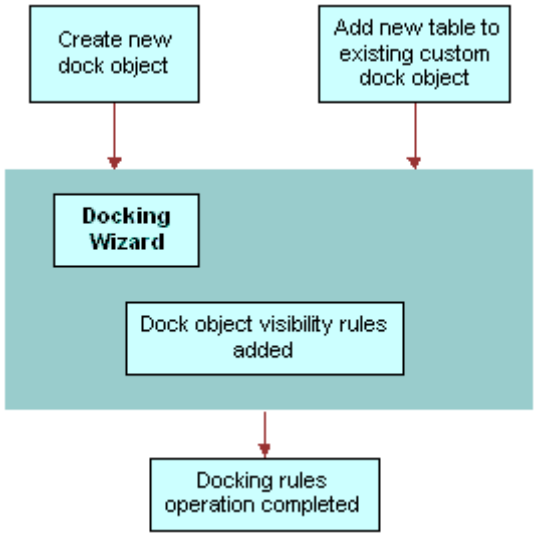


Figure 27. Docking Wizard Flow

Before using the Docking Wizard, you should be aware of the following considerations:

- You can only invoke the Docking Wizard on custom extension tables that are not already placed into any dock objects.
- You cannot invoke the Docking Wizard on standard Siebel out-of-the-box tables. However, you can invoke the Docking Wizard from a custom extension column that has been added to a standard table.
- For a custom table that has a mandatory foreign key to another custom table that is already in a custom dock object, you can either create a new dock object for it or add it to the existing custom dock object. The approach depends on the business requirements and desired outcome.

- The rules created by the Docking Wizard will have a check dock object visibility strength of 100 and a visibility strength of 50. These strengths cannot be modified without assistance from Siebel Systems, Inc.
- When custom tables are added as dock object tables, they are added with a visibility strength of 50. If this is not appropriate for your business requirements, you must seek assistance from Siebel Systems to modify this strength.

To invoke the Docking Wizard from a table

1. Select a custom extension table.
2. Lock the project.
3. Right-click the table record, and then choose Docking Wizard from the pop-up menu.

   The Docking Wizard will not be activated if the table already exists in a dock object. If the Docking Wizard completes successfully for the table, the Docking Wizard will not be activated again on right-clicking the mouse.

To invoke the Docking Wizard from a column

1. Select a custom extension column.

   Custom extension columns have the prefix X_.
2. Lock the project.
3. Right-click the column record, and then choose Docking Wizard from the pop-up menu.

   The Docking Wizard is activated if the column name is prefixed by X_ or the table name is prefixed by CX_ and the table is already in a dock object, whether the table is a Siebel table or a custom table.

   The Docking Wizard can be invoked multiple times, regardless of whether it has been run for this column before or not.

The behavior of the Docking Wizard differs depending on where it is invoked:

- From a table:
  - If the custom table is stand-alone, the only option is to create a new dock object for it. After the dock object creation, appropriate routing rules will also be created.
  - If the custom table has appropriate foreign keys to other custom tables (excluding Siebel tables) already in certain dock objects, there are two options. Appropriate routing rules will be created for either option.
    - Create a new dock object.
    - Add the table to an existing custom dock object.
- From a column:

  You do not need to make any choices. The Docking Wizard will add appropriate routing rules:

- For a regular foreign key, two Check Dock Object routing rules will be added: one from the table's dock object to the foreign key table's dock object and the other in the opposite direction.
- For a foreign key to S_POSTN, only a position rule will be added.

Creating New Dock Objects

If you create a new dock object for a stand-alone customer table, you need to lock the project where you want the new dock object to reside. If you create a new dock object for a non-stand-alone customer table, you not only need to lock the project where you want the new dock object to reside, but also lock all the projects containing the dock objects in which the parent tables of the customer table reside.

If you invoke the Docking Wizard from a stand-alone custom table, only the Create a New Dock Object option is activated. The Add the Table to an Existing Dock Object option will be deactivated. When the Docking Wizard creates rules, it creates rules on associated dock objects. If any other project needs a rule added to a dock object in that project, a dialog box appears warning you that other projects need to be locked, if they are not already locked.

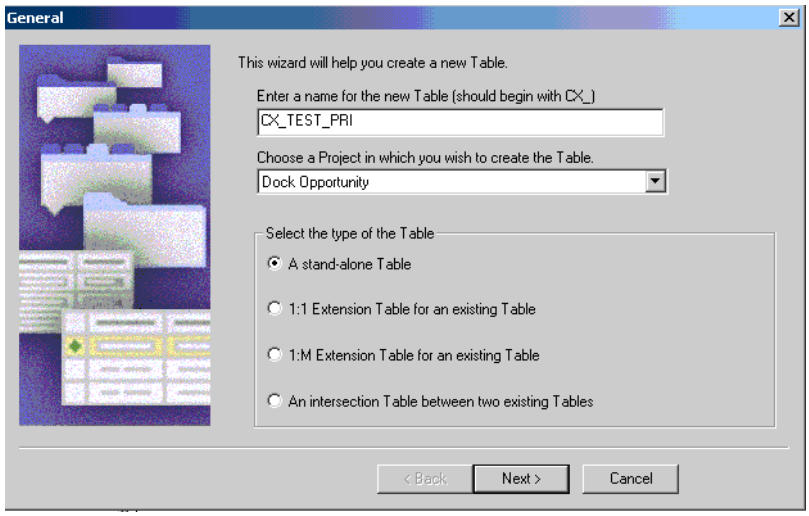
NOTE: You must select a custom extension table (that is, beginning with CX_), and then right-click to access the Docking Wizard.

To create a table for the new dock object

1. Lock the project that will contain your new table, for example Newtable.
2. Select File > New Object.

   The New Object dialog box appears.
3. Select the Table icon under the General tab.

4 In the first General dialog box enter the name of your new table beginning with CX_, select the project, and then select the radio button for the type of table you want.



For example, create a stand-alone custom extension table called CX_TEST_PRI.

5 Click Next.

The Finish dialog box with your entries appears.

6 Click Finish to accept the entries.

You are taken to the Tables Object List Editor, where you see your new table displayed.

To create the new dock object

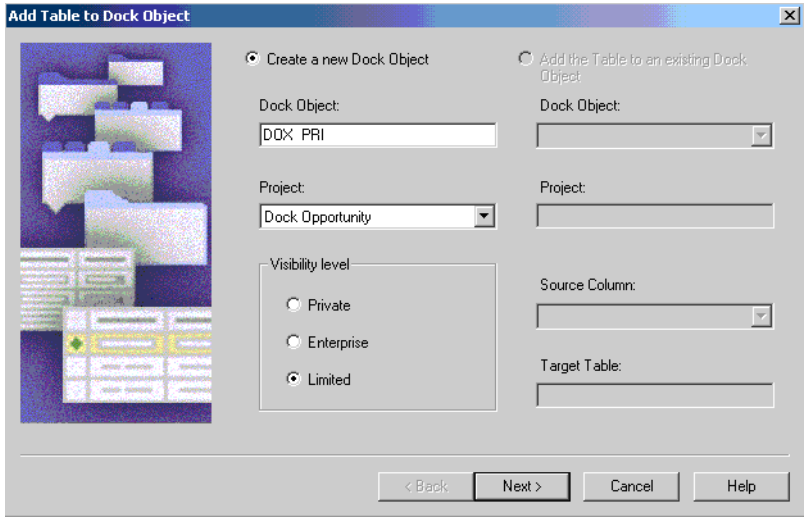
1 Lock the project that will contain the dock object, for example Dock Opportunity.

2 In the Tables Objects List Editor, select the table for which you want to create a dock object (for example, CX_TEST_PRI created above).

3 Right-click, and then select Docking Wizard from the menu options.

NOTE: The Docking Wizard can be launched from entries in the Tables Object List Editor in Table > Column, Table > Index, or Table > User Key object types.

The Add Table to Dock Object dialog box appears.



NOTE: If you invoke the Docking Wizard from a stand-alone table, only the Create a New Dock Object option is activated. The Add the Table to an Existing Dock Object option is deactivated. When the Docking Wizard creates rules, it creates rules on associated dock objects. If a rule needs to be added to a dock object in a different project that is unlocked, a dialog box appears warning you that other projects should also be locked.

4 In the Dock Object field, enter the name of the dock object, for example, DOX PRI.

NOTE: This field must be populated with the DOX prefix.

5 In the Project field, all locked projects are listed in the picklist.

6 Choose the project for the dock object.

7 In the Visibility level section, choose Private, Enterprise, or Limited.

NOTE: If you chose Limited, the employee, employee manager, position, and position manager rules are created on the new dock object, depending on the structure of the table. Dock object rules are created on both the new dock objects and the parent tables' dock objects.

8 Click Next.

The Summary page appears.

9 If the information displayed is correct, click Finish.

The Docking Wizard creates the new dock object.

Adding a New Dock Table to a Dock Object

Your new table should be a dock object table of an existing dock object. In this situation, the new table is a child of an existing dock object table. Mobile users receive records in the new table if they have access to its parent record in the existing table.

NOTE: You must select a custom extension table (that is, beginning with CX_) and right-click to access the Docking Wizard.

You add new dock object visibility rules for existing dock objects when you use the Docking Wizard. By creating a new dock object visibility rule, you provide access to records in existing tables to mobile users who "own" a record in the new table.

This is appropriate when the new table can act as a parent to the primary table of another, limited visibility dock object or when the new table has a foreign key to the primary table of another limited visibility dock object.

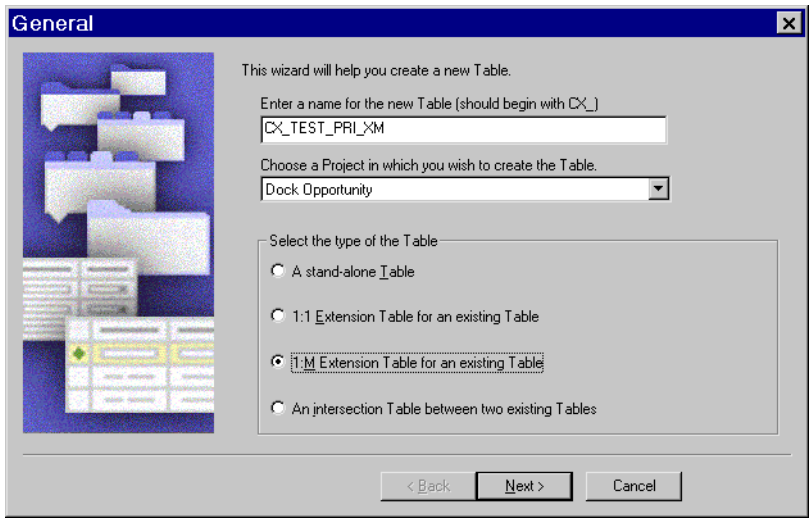
Before launching the Docking Wizard, you need to lock the project containing the customer dock object to which you want to add your new table. You also need to lock all projects containing the dock objects where your new table's parent tables reside.

To create a new table to be added to an existing dock object

1. Lock all necessary projects.
2. Select File > New Object.

   The New Object dialog box appears.
3. Select the Table icon under the General tab.

4 In the first General dialog box enter the name of your new table beginning with CX_, select the project, and then select the radio button for the type of table you want.



For example, create a new one-to-many extension table of CX_TEST_PRI.

5 Click Next.

The Parent Table Specification dialog box appears.

6 Specify the parent table, for example CX_TEST_PRI.

7 Click Next to display the Finish dialog box with a summary of your choices.

8 Click Finish.

The new table is created and displayed in the Object List Editor.

To add the new table to an existing dock object

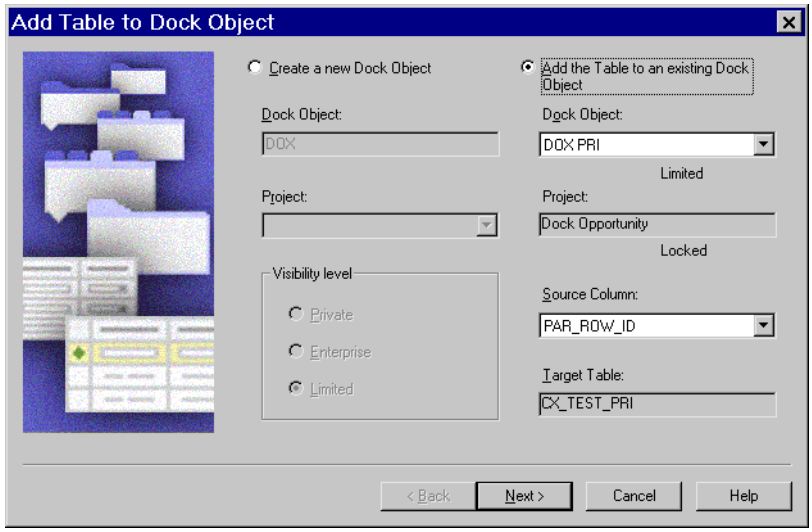
1 Select the Table object type in the Object Explorer.

2 Select the new table, for example CX_TEST_PRI_XM, in the Tables list.

3 Right-click and select Docking Wizard from the menu options.

The Add Table to Dock Object dialog box appears.

4 Select the Add the Table to an Existing Dock Object radio button.



5 Select an entry in the Dock Object field.

The choices are a list of all Dock Objects that contain tables to which the new table has a foreign key.

The associated locked project is displayed in the Project field.

6 Select an entry in the Source Column field.

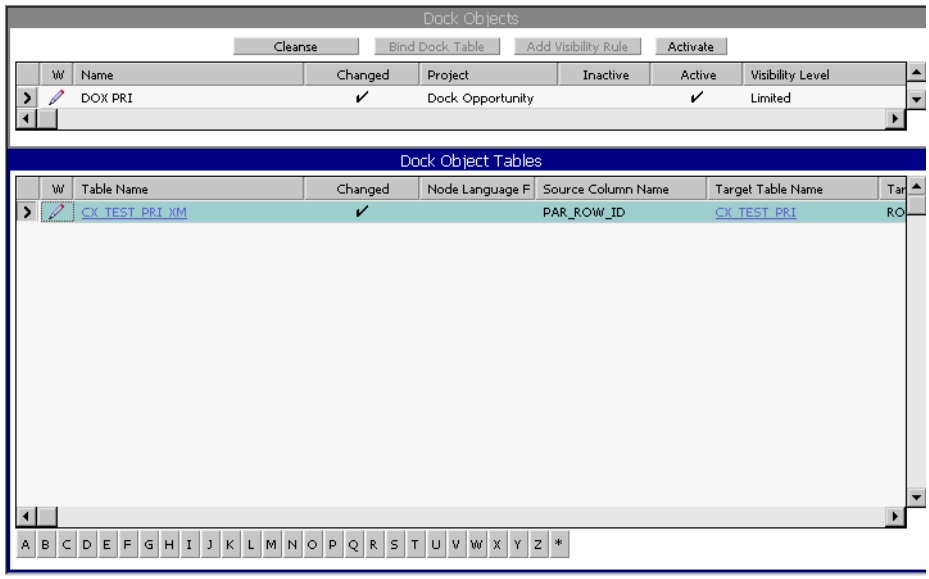
This field allows you to choose a column from the new table that is a foreign key to the parent table contained in the selected Dock Object Table. Frequently, there is only one such column, but there may be more in some cases.

NOTE: When the Source Column field entry is selected, the Target Table field is populated.

7 Click Next to display the Summary dialog box.

8 If the information displayed is correct, click Finish.

The Docking Wizard creates a Dock Object Table object based on the new table and displays it in the Object List Editor.



Verifying Dock Objects

You can verify newly created dock objects, dock object tables, and dock object visibility rules within Siebel Tools.

To verify dock objects

1 In the Object Explorer, select Dock Object.

2 Expand Dock Object, and then select Dock Object Table.

3 Look for the created item in the Dock Object Tables list.

4 Select Dock Object Visibility Rule.

5 Look for the created item in the Dock Object Visibility Rules list.

For more information on dock object visibility rules, see "Dock Object Visibility Rules" on page 152 and *Siebel Remote and Replication Manager Administration Guide*.

Deleting and Cleansing Dock Objects

Custom dock objects (those with prefix DOX) can be deleted.

When a custom table, column, or dock object is deleted, or when a foreign key column is redefined to point to a different table, dock object integrity may be broken. The dock objects will need to be cleansed before the Docking Wizard is used again or before Siebel Remote can be used.

The Cleanse button is located on the Dock Object list applet in Siebel Tools. By clicking it, all dock objects are examined, and you are prompted to make sure the dock objects are all clean; if not, some objects will be deleted. If the projects on which you are working are not locked, you are prompted to lock them. After the process is completed, you are prompted again on what has been deleted.

Custom dock objects (those with prefix DOX) can be deleted.

When a custom table, column, or dock object is deleted, or when a foreign key column is redefined to point to a different table, dock object integrity may be broken. The dock objects will need to be cleansed before the Docking Wizard is used again or before Siebel Remote can be used.

The Cleanse button is located on the Dock Object list applet in Siebel Tools. By clicking it, all dock objects are examined, and you are prompted to make sure the dock objects are all clean; if not, some objects will be deleted. If the projects on which you are working are not locked, you are prompted to lock them. After the process is completed, you are prompted again on what has been deleted.

7 Configuring Business Components

Topics in This Chapter

About Business Components

A *business component* is a logical representation of one or more tables. Business components provide the foundation for controlling how data is selected, inserted, and updated in underlying tables.

The information stored in a business component is usually specific to a particular functional area, such as a product, a contact, or an account. This information may or may not depend on other business components. Business components can be included in one or more business object definitions. They can have default sort or search specifications that allow you to expose records in the user interface in a predetermined sort order and according to a set of selection criteria. Multiple users can instantiate copies of the same business component. Data changes made by any one user are reflected in all instances of the business component.

The main data for a business component comes from a base table and one or more joined extension tables. For example, the Account business component is based on the S_PARTY table, but most of the data retrieved by the business component is stored in the joined extension table, S_ORG_EXT.

Business Components Include Data From Base Tables

Every business component, except virtual business components, has a base table assigned to it. For non-party business components the *base table* provides the most important columns for use as fields in the business component. The base table is assigned to the business component with the Table property in the Business Component's object definition. Fields in the business component map to columns in the base table.

Figure 28 shows an example of fields in the Contact business component that map to corresponding columns from the business component's base table, S_CONTACT.

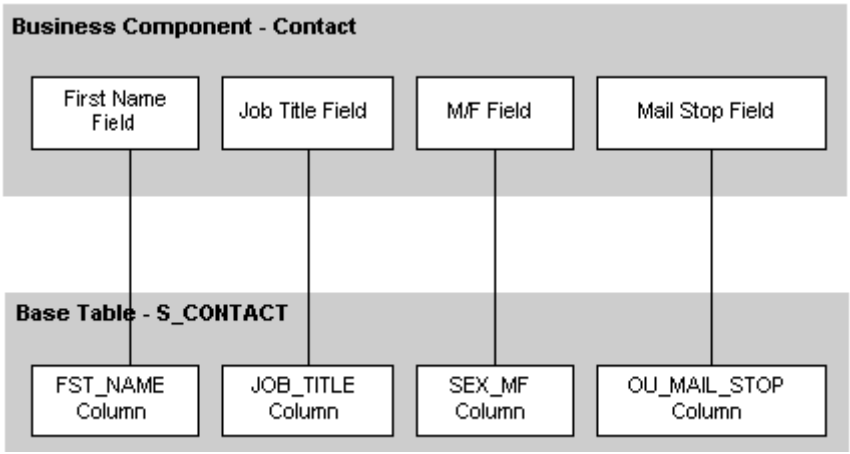


Figure 28. Examples of Fields Representing Columns

NOTE: Business components do not need to include all of the columns in the base table, although typically it will include most of them. In particular, system columns in the base table such as ROW_ID, CREATED_BY and LAST_UPD_BY are automatically represented in the business component through implied fields. System columns do not require field object definitions in the business component.

Business Components Can Include Data from Joined Tables

In addition to including data from base tables, business components can also include data from extension tables and joined tables. For party-based business components, the main data for the business components comes from a joined table. The relationship between the business component and the additional table is defined using a join.

- For extension tables, the join is defined implicitly when the extension table is created. Fields that map to columns in extension tables can typically be edited by users. For more information, see "About Extension Tables" on page 92.
- For other tables, the join is defined explicitly as a Join object using Siebel Tools. Fields based on columns from joined tables are typically for display only. For more information, see "About Joins" on page 199.

A *joined table* provides rows on a one-to-one basis to the business component as a result of a foreign key relationship between the joined table and the business component's base table. That is, for every record in the business component (which corresponds to a row in the base table) there can be a corresponding row in the joined table. However, not every record in the base table will have a record in the joined table.

Figure 29 shows fields in a business component mapping to columns in a base table and a joined table.

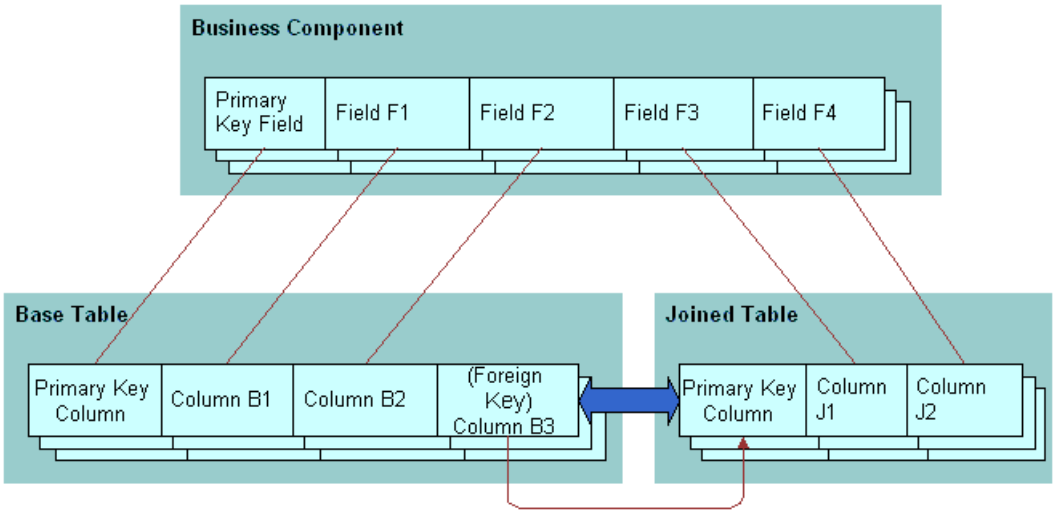


Figure 29. Fields from the Base Table and a Joined Table

Business Components Can Be Reused

Business components can be defined once in terms of a logical collection of columns from one or more tables, and then used in many different business object contexts, as shown in Figure 30.

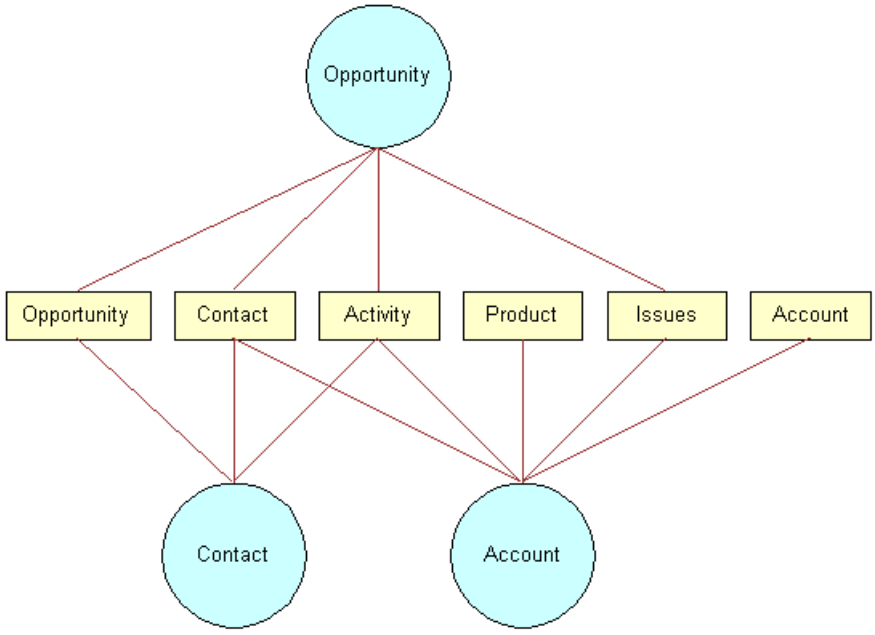


Figure 30. Business Component Reuse

About Virtual Business Components

Business components based on external data are called virtual business components. Virtual business components are used when the business component has to obtain data from a location other than a database table in the Siebel database, but the information has to be presented in the standard Siebel user interface (applets and views). This is typically real-time information from another database, such as from the Report Encyclopedia in Actuate, or from an SAP table, although anything that can supply data in response to an SQL query is a candidate.

Virtual business components allow you to:

- Represent external data (for example, data in an SAP R/3 database) as a virtual business component within a Siebel application—the business component configuration specifies the DLL to use to access the data
- Use business services to transfer data

Virtual business components support properties such as:

- Single-value fields
- Field-level validation
- Standard business component event model (for example, PreNewRecord, PreDelete, and so on)

- Insert, delete, query, and update operations

Additional information about virtual business components:

- Applets can be based on virtual business components.
- Virtual business components can be accessed through object interfaces.
- All business component events are available for scripting.
- Virtual business components cannot be docked.
- Virtual business components can be used as stand-alone or children business components in a business object.
- Virtual business components support dynamic applet toggles.

  For more information about applet toggles, see "About Applet Toggles" on page 297.
- Virtual business components can function as parent member of Link objects in 1:M relationships with standard business components.
  - Virtual business components generate Siebel row IDs the same way as standard business components.
  - M:M relationships involving virtual business components are not supported.
  - M:M relationships in an external system (for example, SAP) that function similarly to a 1:M relationship are supported.

For more information about using virtual business components, see *Overview: Siebel eBusiness Application Integration Volume I*.

Guidelines for Configuring Business Components

Often, during configuration, there is a need for a business component that is similar, but not identical, to an existing business component. In this case, you must choose between creating an entirely new business component based on the existing one or modifying the existing one for reuse. Modifying an existing business component is usually the better solution, because it minimizes the number of business components. This leads to a smaller repository, is easier to maintain, and is easier to upgrade (because it is closer to standard applications).

Always configure your application in a way that lets you reuse business components instead of creating new ones. For example, you can have an implementation where one group of users may create opportunities, but another group can only edit existing opportunities. Instead of creating a new business component and setting the No Insert property to TRUE, you can define a new applet and set the No Insert property to TRUE for the applet.

If you must create a new business component, avoid copying business components based on a specialized class, unless you intend to create a true clone of the original business component (with the same functionality), and then apply minimal changes. For example, you may want to create a Locked Service Requests business component that displays only those Service Request records that have been locked using a business component User Property. In this example, you would copy the Service Request business component (defined by the specialized class CSSBCServiceRequest), set up the Lock Field business component User Property, and specify the conditions in which a Service Request should be locked. Next, you would identify a search specification for the business component that will retrieve only those Service Request records with the preceding conditions. The underlying behavior of the new business component remains the same as the original business component. You should avoid copying a specialized business component to reproduce an isolated feature associated with that business component.

- When naming currency code and exchange date fields, call them Currency Code and Exchange Date if they are the only such fields in the business component. If there are multiple instances of similar fields, prefix each with the name of the corresponding Amount column—for example, Revenue Currency Code and Budget Currency Code. The reason for this is that they are referenced by other fields when you specify the Properties Currency Code field and Exchange Date field. Defining the fields this way makes the reference easier to understand.
- The field URL must be named URL and the class of the Business Component must be set to CSSBCBase for the hyperlinking functionality to work correctly.
- Business components used to represent child entities should not have their parent entity in their name—for example, ABC Subsegment instead of ABC Account Subsegment. Similarly, the applets that are based on these child business components should only reflect the name of the business component itself—for example, ABC Subsegment List Applet instead of ABC Account Subsegment List Applet.

  NOTE: The exception is when you need multiple variations of the same business component or applet. Typically, multiple variations are necessary if a particular entity is displayed as both a top-level applet and a child applet on other views, and the two applets are not the same. In such cases, put the name of the parent entity at the beginning of the child applet name. For example, the ABC Account Contact List Applet is a Contact List that is displayed as the child of an Account. It needs the word Account to distinguish itself from the standard ABC Contact List Applet, which is a different applet.

Related Topics

"About Object Reuse" on page 51

"About Reusing Business Components" on page 58.

"About Copying Objects" on page 48

Creating Business Components

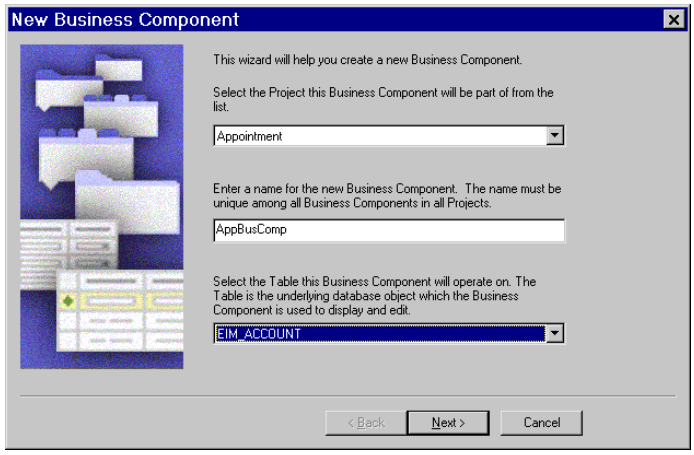
You use the New Business Component Wizard to create Business Component objects. You may need to create new business components when existing business components are not a good functional or technical fit to your business requirements. For example, suppose you need to define scripting logic that is different from any similar, out-of-the-box business components or suppose you want to predefault record values to a different type in order to differentiate records from other records in the same table. In both cases, you would create a new business component.

For more information on reusing existing objects and creating new objects, see "About Object Reuse" on page 51.

To open the Business Component wizard

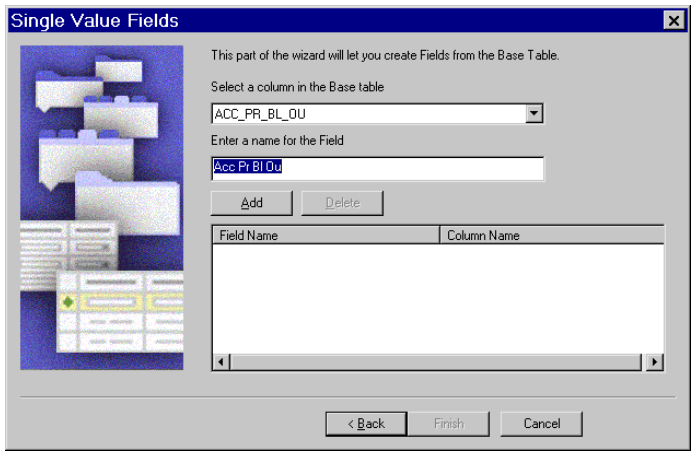
1. Choose File > New Objects.
2. Select the BusComp icon, and click OK.

   The New Business Component dialog box appears as shown in the following figure.



3 Select a Project and the master Business Component and click Next.

The Single Value Fields dialog box appears.



4 Select a column in the Base table and enter a name for the field.

5 Click Add and then click Finish.

When you click Finish, you are taken to the business component you just created in the Object List Editor, where you can further configure the new Business Component object.

For more information about business components, see "About Business Components" on page 167.

Defining Business Component Properties

Following are descriptions for key properties for the Business Component object type. For a complete list and more detailed descriptions, see *Object Types Reference*.

- **Class.** The C++ class that implements the functionality of the business component. New stand-alone business components should have a class property of CSSBusComp or CSSBCBase. A drop-down list of values appears for setting this property.

  Siebel applications have a hierarchy of business component classes. CSSBusComp is at the top of the hierarchy. All other specialized business component classes (CSSBCOppty is an example) are derived from CSSBusComp.

  The functionality that is common between business components includes navigation (moving through a result set returned from the database), get or set field values in records, create and delete records, commit changes, undo/redo, bookmark, search, and sort.

  NOTE: Do not change the Class property of preconfigured business components. When creating a business component by copying an existing one, do not change the class property.

  For more information about classes, see *Siebel Developer's Reference*.

- **Name.** (Required.) Must be unique among all business components in the repository. All references to the business component are done through its name.
- **No Delete, No Insert, No Update properties.** (Default is FALSE.) If set to TRUE, then you cannot do data manipulation operations.
- **Search Specification.** A conditional expression used to restrict the records retrieved.
- **Sort Specification.** A sort expression used to order the records returned.
- **Table.** (Required.) The name of the SQL table from which records are retrieved to populate the majority of fields in the business component. A list of tables appears in a picklist.

NOTE: You can configure queries to be case sensitive or case insensitive by setting the Use Default Sensitivity property in the configuration (.cfg) file. Set the property to TRUE for case-sensitive queries. Set the property to FALSE for case-insensitive queries. However, you cannot configure fields of the type DTYPE_ID in this way because these fields always conduct case sensitive searches.

Defining Sort Specifications

The value in the Sort Specification property, if it is non-blank, is the name of a field or list of fields that imposes a sort order on the records returned to an applet that is associated with this business component. The field or fields must be child object definitions of the business component.

For example, the Account business component (as delivered in Siebel applications) has a Sort Specification property value of "Name, Location." This indicates that account records are provided in Name (account name) order, and where multiple account records have the same Name, they are to be sorted within Name by Account Location.

Observe the following syntax considerations:

- Use commas to separate field names in a sort specification.
- To indicate that a field in the list sorts in descending order, include (DESCENDING) or (DESC) after the field name, as in "Start Date (DESCENDING)." If you do not specify a sort order, ascending order is used.
- Do not enclose the field name in square brackets, as in [Account Name]. Brackets are accepted in search specifications, but not in sort specifications.
- The sort specification expression must be 255 characters or less.

Be aware that sort specifications have the following behaviors:

- If the Sort Specification value is blank, the Siebel application returns the records in the order in which they occur in the table.
- When a check box field is included in a sort specification, there are three values that are sorted: Y, N, and NULL. If you specify that the sorting is in Descending order, the order is NULL, Y, and N.
- When a multi-value field is included in a sort specification expression in a business component, the sorting is on the initial value of the multi-value field. This makes sense only if the multi-value group uses a primary foreign key.

- A sort specification that includes a multi-value field in the expression does not sort the records in the underlying multi-value group. Instead, you create a sort specification in the detail business component of the multi-value link to do this.
- For sorting the values in a static picklist or pick applet differently than the default sorting for the underlying business component, the sort specification on the business component can be overridden with a sort specification on the picklist. The default value for the Sort Specification property in a Pick List object definition is blank, which means that the business component's sorting is to be used. If a sort specification appears in the picklist, this overrides the business component's sorting with that of the picklist.
- You cannot sort on a calculated field.
- If a predefined query exists, it can potentially override a sort specification that has been defined as a property of the business component.
- Sort specifications can have a negative effect on performance. This is particularly true when the sorting is on fields based on joins or on fields based on new, non-indexed extension columns. For more information, see *Performance Tuning Guide*.
- Siebel applications force the sort in the All visibility mode to be on the primary key. The sort in Manager mode is on a column in the denormalized reporting relationship table. You can still sort records after the initial query. For better performance, you should sort records after filtering for a small record set.

  NOTE: You can use the All Mode Sort business component user property to force the application to use custom sort specifications or to ignore all sort specifications. For more information, see *Siebel Developer's Reference*.
- Null records will always appear at the top of the record set if a sort specification is placed on a field with null values.

Defining Search Specification Property

If the value in the Search Specification property in a Business Component object definition is non-blank, the set of records provided to an applet using this business component is restricted. The search specification contains the names of one or more fields in the business component and various operators, combined to create a conditional expression. Records in which the value of the conditional expression evaluates to TRUE are provided to the applet for display; those records in which the expression evaluates to FALSE are excluded.

NOTE: Search specifications on child applets are not executed.

Some sample search specification expressions appear below:

```
[Type]= "COST LIST"

[Revenue] > 5000

[Competitor] IS NOT NULL and [Competitor] <>  "N"

[Type] = LookupValue ("TODO_TYPE", "In Store Visit")
```

Search specification expressions are built according to the following syntax rules:

- Standard comparison operators are used to compare a field to a constant, or one field to another field. These include =, <>, >, <, >=, and <=.

  Example: [Revenue] > 5000

- String constants are enclosed in double quotation marks. String values are case sensitive, so the use of uppercase and lowercase letters in the search specification should exactly match that of the records you want returned.

  Example: `[Type] <> "COST LIST"`

- The logical operators AND, OR, and NOT are used to negate or combine expressions. Case is ignored in these operators; for example, "and" is the same as "AND").

  Example: `[Competitor] IS NOT NULL and [Competitor] <>  "N"`

- A field name in a search specification must be enclosed in square brackets.

  Example: `[Conflict Id] = 0`

- The search specification expression must be 255 characters or less.

An applet search specification cannot be used to override the search specification of the underlying business component, if the business component has one. Rather than overriding the business component's search specification, the applet's search specification is appended to that of the business component. Search specifications should appear in the business component or the applets that use it, but not both.

The search specification on an applet is converted to a WHERE clause by the data manager at run time. When two applets based on the same business component appear in the same view, one query is generated against the database to populate both applets. Because a database select statement only supports one WHERE clause, only one of the applets should have a search specification—or if both do, they should have the same specification.

For example, the Account List Applet and the Account Entry Applet both appear in the Account List View. The record that is selected in the Account List Applet also appears in the Account Entry Applet. When you select a different row in the list or scroll through the list, the Account Entry Applet is updated to show the same record that is selected in the Account List Applet. This is made possible by the fact that both applets are populated from the same query and therefore show the same record set.

To prevent the two applets from being synchronized, they would have to be on separate business components, for example by copying the business component on which the first applet is based.

For more information on the usage of the Search Specification property of applets, see *Siebel Developer's Reference*.

Search specifications can affect performance negatively, particularly when you include:

- Fields based on joins
- The operators NOT or OR; they can force the database to execute a full table scan
- Calculated fields

For more information about performance, see *Performance Tuning Guide*.

About Fields

Fields are child objects of business components. They are the source of data for controls and list columns displayed on applets in the user interface. Typically fields represent:

- Information from a database column obtained through a column object definition. Columns can be from the base table, extension tables, and joined tables of the business component.
- A calculated value that is derived from the values in other fields, but not stored in the database. These fields are knows as calculated fields. For more information, see "About Calculated Fields" on page 186.

Figure 31 illustrates data from fields displayed in a form applet.

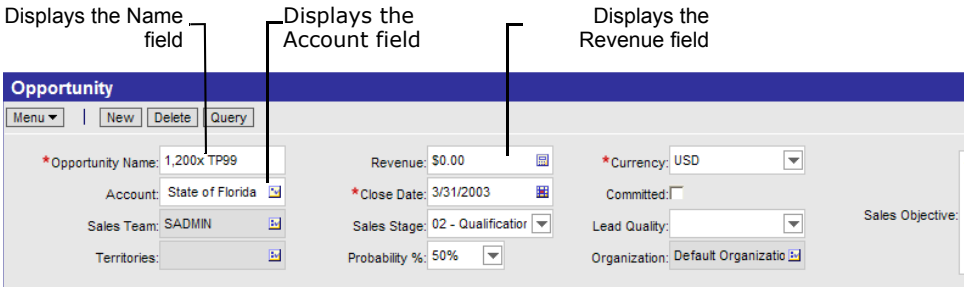


Figure 31. Data from Fields Displayed in a Form Applet

Figure 32 illustrates data from fields displayed in a list applet.

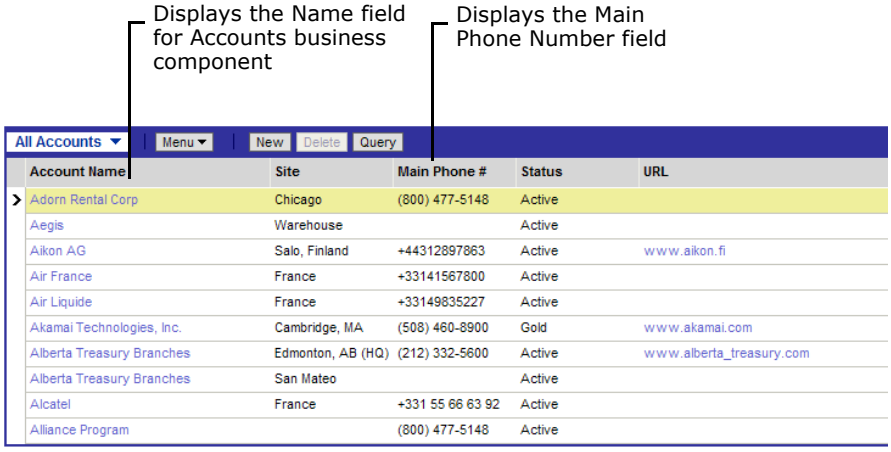


Figure 32. Fields Displayed in a List Applet

Figure 31 and Figure 32 show controls in a form applet and list columns in a list applet that obtain their data from fields in a business component. The Field property setting in a Control or List Column object definition specifies the field. The Business Component property in the applet specifies the business component. These property relationships are illustrated in Figure 33.

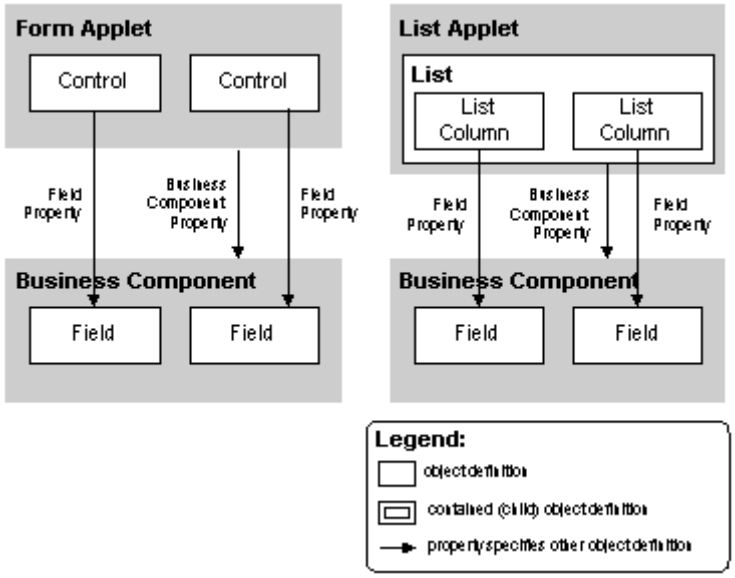


Figure 33. Field Property Relationships

In field object definitions that are not calculated fields, the column and Join properties together specify the table and column from which the field's data is obtained. The Join property, if blank, indicates that the column is obtained from the business component's base table. If it is nonblank, the Join property identifies the join object definition that supplies data from an extension table or other joined table. Based on the Join property, the table supplying the field's data is identified. The Column property identifies the column to use within the specified table. These relationships are illustrated in Figure 34.

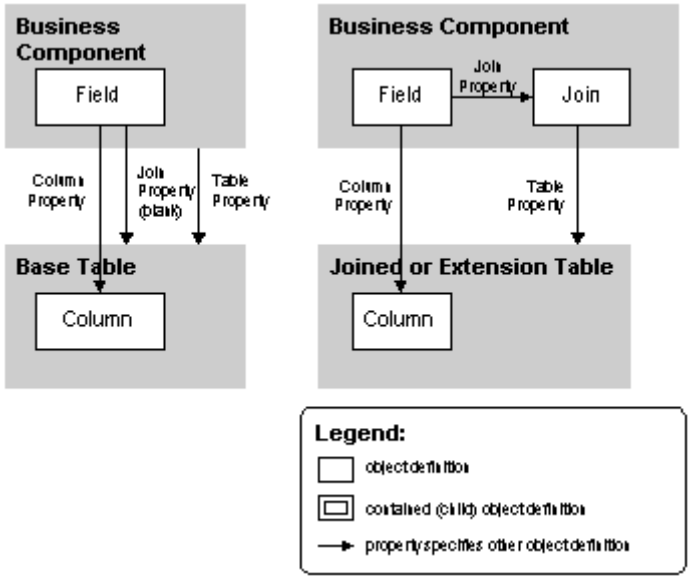


Figure 34. Field Relationship Details

NOTE: You should not map multiple fields to the same column in a table. This can lead to error messages when updating the data and can cause data integrity issues. The SQL query fails because it tries to access the same column twice in the same query. It can also lead to data loss for denormalized columns that reference the column.

About Field Data Types

The Type property specifies the data type for the field. Field data types are used to identify the type of data retrieved from and sent to the database server.

These data types are not mapped to the physical data types defined by the database. The data type of the field is generally more specific than the data type of the underlying column (as identified in the Physical Type property of the column). For example, both DTYPE_NUMBER (decimal) and DTYPE_INTEGER field data types have the Number physical data type in the column.

It is not recommended to map a field to a different table column type, for example a DTYPE_NUMBER field mapping to a table column of type Varchar.

Just as the data type of the underlying column restricts the set of field data types that will work correctly, the data type of the field restricts the set of correctly functioning format options in the control or list column that displays it.

Most formatting is defaulted from the Microsoft Windows Control Panel. Overriding the default format in the repository is possible but might lead to confusion. For example, overriding a number format to show more or fewer decimal places would be useful, but overriding a date format to DD/MM/YY would be confusing to a user who has set the date format to MM/DD/YY in the Control Panel.

NOTE: Multi-value fields (fields with a Multi Valued property setting of TRUE) have a blank Type property, because the data type of the field is specified in the detail business component that populates it.

All field data types are prefaced with DTYPE_.

Table 36. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_BOOL	Character	1	Refers to data stored as Y or N, often displayed as TRUE or FALSE and checked or unchecked.
DTYPE_CURRENCY	Number	22	Refers to the data as currency. You can control the appearance of currency values on a screen through the Windows Control Panel, or you can specify an explicit format mask in the Display Format property by using the following symbols: ■ Dollar sign ($). Specifies the position for the currency symbol. ■ Trailing period (.). Specifies the default precision for the currency. ■ All valid symbols described for DTYPE_NUMBER.
DTYPE_DATE	Date	7	Refers to the data as a date. When the date is returned, any additional information (for example, time) is ignored. You can set the appearance of date values through the Windows Control Panel, or you can specify an explicit date format using the following symbols: ■ YY. Two-digit year without a leading zero. ■ Y. Two-digit year with a leading zero. ■ YYYY. Four-digit year without a leading zero. ■ YYY. Four-digit year with a leading zero. ■ MM. Month without a leading zero. ■ M. Month with a leading zero. ■ DD. Day without a leading zero.

Table 36. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_DATETIME	Date Time	7	Refers to the data as a date and time. You can set the appearance of time and date values through the Windows Control Panel, or you can specify an explicit date format using a combination of the symbols for DTYPE_DATE and DTYPE_TIME. Alternatively, you can use one of the following properties: ■ Date. Displays only the date portion of the value, using the format specified in the Windows Control Panel. ■ Time. Displays only the time portion of the value, using the format specified in the Windows Control PanelTimeNoSec. Displays only the hour-and-minute portion of the value, using the format specified in the Windows Control Panel.
DTYPE_ UTCDATETIME	UTC Date Time	30	Indicates that the corresponding field represents date information, with both a date and a time component, that will be stored in the database in UTC time (UTC is the equivalent of Greenwich Mean Time without any adjustments for daylight savings time). Fields of this type should correspond to database columns of type *U*, and the display values for these fields will be converted to/from UTC based on the default time zone specified in the user's preferences.
DTYPE_ID	Varchar	15	Refers to the data as the primary key automatically generated by the application. Fields mapped to extension columns of physical type Varchar(15) will automatically default to data type DTYPE_ID.
DTYPE_INTEGER	Number	22*	Refers to data as whole numbers ranging in value from - 2147483648 to 2147483647.

Table 36. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_NOTE	Long	16 KB	Refers to the data as a long string less than or equal to 16 KB (16383 bytes); the default, if the length is not explicitly defined, is 16 KB. When used with the Pop-up Edit property in a control or list column, this data type is used to indicate to the user interface that a multiline edit box should be used. Users cannot query on fields of type DTYPE_NOTE.
DTYPE_NUMBER	Number	22	Refers to the data as a number. You can control the appearance of numeric values through the Windows Control Panel, or you can specify an explicit format mask using the following symbols: ■ Zero (0). Specifies the position of a mandatory digit. ■ Pound sign (#). Specifies the position of an optional digit. ■ Comma (,). Specifies the position of the thousands separator (you specify the character in the Windows Control Panel). ■ Period (.). Specifies the position of the decimal separator (you specify the character in the Windows Control Panel).
DTYPE_PHONE	Number	40	Refers to the data as a phone number. The DisplayFormat property is ignored for values of this type.

Table 36. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_TEXT	Varchar	2 KB	Refers to the data as a string less than or equal to 2000 bytes; the default is 255. The DisplayFormat property is ignored for values of this type. You can use ForceCase = "Upper" or ForceCase = "Lower" to force the text to all uppercase or all lowercase after the end user tabs out of the field. You can use ForceCase = "FirstUpper" to force the first letter of each word to uppercase after the user steps off the record. Otherwise, the text is in mixed case as the user entered it.
DTYPE_TIME	Time	7	Refers to the data as a time. When the time is retrieved, any additional information (such as date) is ignored. You can set the appearance of time values through the Windows Control Panel, or you can specify an explicit time format using the following symbols: ■ HH. Hour (based on 24-hour clock) without a leading zero. ■ H. Hour (based on 24-hour clock) with a leading zero. ■ hh. Hour (based on 12-hour clock) without a leading zero. ■ h. Hour (based on 12-hour clock) with a leading zero. ■ mm. Minute without a leading zero. ■ m. Minute with a leading zero. ■ ss. Second without a leading zero. ■ s. Second with a leading zero. ■ Colon (:). The position of the time separator (you specify the character in the Windows Control Panel).

About System Fields

System fields are provided in all business components in standard Siebel eBusiness Applications. These fields represent the data from system columns.

Table 37 identifies the correspondences between system fields and system columns.

Table 37. System Fields and Their System Columns

System Field Name	System Column Name	Description
Id (or blank)	ROW_ID	Primary key for the table.
Created	CREATED	Creation date and time of the row.
Created By	CREATED_BY	User logon ID of the person who created the row.
Updated	LAST_UPD	Date of last update of the row. Updated (system field) is updated only when the row with the Updated column in it is changed.
Updated By	LAST_UPD_BY	User logon ID of the person who last updated the row.

System fields are automatically available to expose or manipulate in business components. You do not need to define them as business component fields. For example, system fields can be referenced in the Field property of controls, list columns, and other object definitions.

NOTE: Do not change system fields, for example by renaming them. Changing Siebel system fields is not supported.

About Calculated Fields

Calculated fields have a Calculated property of TRUE and a non-null value for the Calculated Value property. Calculated fields obtain their values from other fields in the same business component, or from the master business component in an active link in which the current business component is the detail.

The Calculated Value property contains an expression built from field names, standard functions, and string, numeric, and logical operators. For example, the Full Name field in the Contact business component has the following Calculated Value property setting:

```
IIf (Language () = "JPN", [Last Name] + ' ' + [First Name],
[First Name] + ' ' + [Last Name])
```

The meaning of this expression is as follows: if the active client language setting is Japanese, construct the Full Name from the Last Name, a blank space, and then the First Name. Otherwise, construct the Full Name from the First Name, a blank space, and then the Last Name.

When configuring calculated fields, consider the following:

- Calculated fields are not automatically refreshed when a related field value changes; they are refreshed only after committing the record. To have them refresh immediately after the fields have been changed the Immediate Post Changes property of the field needs to be set to TRUE.
- A calculated field cannot reference itself in the Calculated Value property. For example, you cannot use [Last Name] in a calculation expression for the Last Name field.
- Queries on calculated fields are not supported if the Cache Data property of the business component is set to TRUE.

For information on the construction of calculated field expressions for the Calculated Value property, see *Siebel Developer's Reference*.

Creating Sequence Fields

Situations can occur in which you need to create a field that provides sequential numbering for the parent business component. For example, you may need to number line items in an Order or products in an Opportunity. Sequential numbering is not automatically provided in any system columns in standard tables in Siebel applications. However, you can configure a sequence field in a detail business component by adding a business component user property called Sequence Field and creating a sequence business component with a special business component class called CSSSequence.

The details of configuration of a sequence field appear in Figure 35.

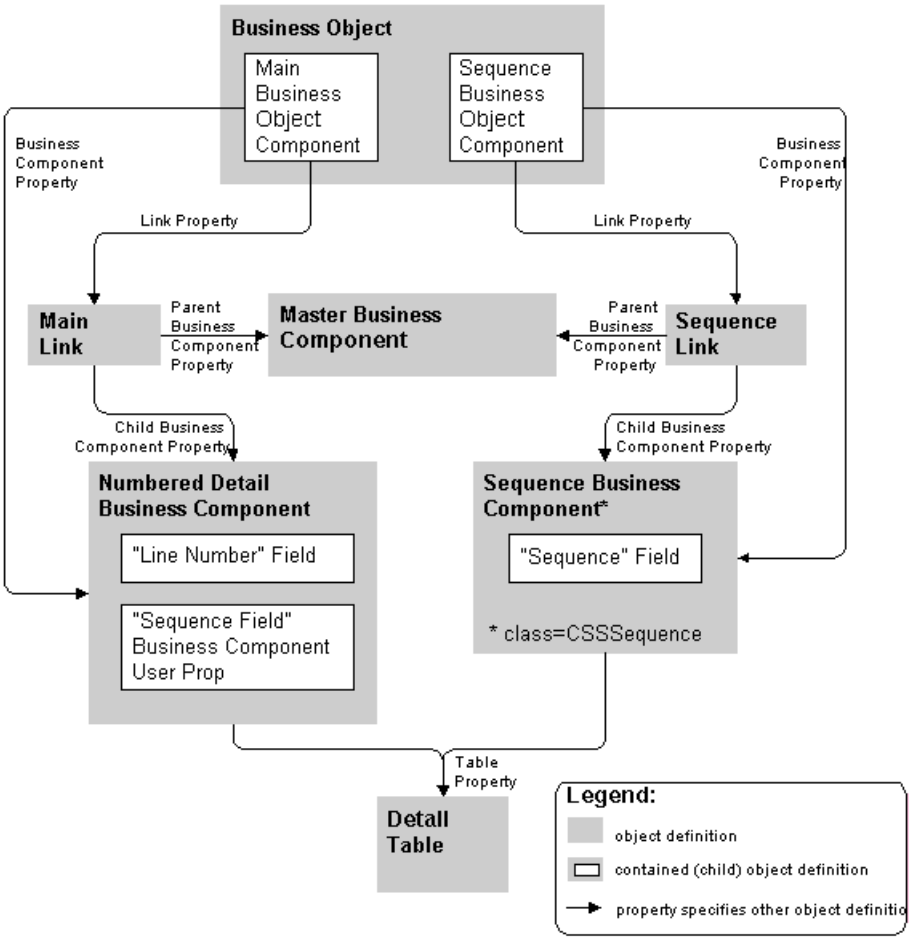


Figure 35. Sequence Field Configuration Details

The roles of the object definitions in Figure 35 on page 188 are as follows:

- **Master business component.** In the master-detail relationship in which the detail records are to be numbered, this is the business component that holds master records. For example, the Opportunity business component is the master in the master-detail relationship with Opportunity Product.
- **Numbered detail business component.** In the master-detail relationship, this is the business component that holds detail records. For example, the Opportunity Product business component is the detail in the master-detail relationship with Opportunity. The numbered detail business component has the following important child object definitions:
  - **Line Number field.** This field, named Line Number, is a field of type DTYPE_NUMBER that holds the resulting sequence value.
  - **Business component user.** A business component user property object definition named Sequence Field needs to be present, with the Value property set to "Line Number."
- **Sequence business component.** This business component is named *xx*.Line Number (Sequence), where *xx* is the name of the numbered detail business component. It has a specialized class of CSSSequence, and the following two fields:
  - **Sequence field.** This field, named Sequence, is of type DTYPE_NUMBER.
  - **Foreign key field.** This field is a foreign key field based on a foreign key column in the detail table. The foreign key column points to the primary key of the base table of the master business component, and may be used to specify the link between the master and sequence business components.
- **Detail table.** The detail table is the base table for both the numbered detail and sequence business components.
- **Links.** One link provides the master-detail relationship between the master and numbered detail business components. The other link provides the master-detail relationship between the master and sequence business components. The link to the numbered detail business component is usually preexisting, such as Opportunity or Opportunity Product. The link to the sequence business component is usually added by the developer, except when the sequence configuration is included in standard Siebel applications. Opportunity or Opportunity Product.Line Number (Sequence) is an example of a link to a sequence business component.
- **Business Object.** The second link is included in the same business object that holds the first link.

Examples of sequence field configuration in standard Siebel applications can be viewed in Siebel Tools. For example, examine the Opportunity Product and Opportunity Product.Line Number (Sequence) business components, which are the numbered detail and sequence business components, respectively.

To add a sequence field to a business component that does not currently have one

1. Verify that the class of the detail business component is CSSBCBase or a sub-class of CSSBCBase.

   If it is not, contact Siebel Technical Services for assistance with this procedure.

2. Verify that the business component to which you want to add a sequence field is the detail business component in a master-detail relationship. This is the numbered detail business component.

   NOTE: The numbering of detail records will start from 1 within each master record.

3. Add a child field object definition to the numbered detail business component. Set the Name property value to Line Number, the Type to DTYPE_NUMBER and the column to a numeric extension column such as ATTRIB_14.
4. Add a child business component user prop object definition to the numbered detail business component. Set the Name property value to Sequence Field, and the Value property to Line Number.
5. Create a business component. Set the Class property to CSSSequence, the table to the name of the base table of the numbered detail business component, and the Name to *xx*.Line Number (Sequence), where *xx* is the name of the numbered detail business component. This is the sequence business component.
6. Set the Sort Spec of the *xx*.Line Number (Sequence) business component to Sequence (DESCENDING).
7. Add a child field object definition to the sequence business component. Specify a Name property value of Sequence, and a column value the same as the extension column specified for the Line Number field in the numbered detail business component.
8. Add a child field object definition to the sequence business component. This is the foreign key field that establishes the master-detail relationship to the master business component. The Column property should be set to the same column as the corresponding field in the numbered detail business component.
9. Create a link object definition that establishes a master-detail relationship between the master and sequence business components.
10. Create a Business Object Component child object definition of the business object or business objects that use the existing link between the master and numbered detail business components. Specify the new link and the sequence business component in the Link and Business Component properties, respectively.
11. Expose the Line Number field in applets that display records from the numbered detail business component.

Configuring Data-Driven Read-Only Behavior

Business components and fields can be configured as dynamically accessible, with their read-only status turned on and off depending on the value in a particular field in the current record. This is accomplished using one of the following Business Component object type user properties, depending on the requirement:

- **BC Read Only Field**

  Specifies a TRUE or FALSE field in the record that, when TRUE, causes the current record to become read-only.

- **Field Read Only Field:** *fieldname*

  Specifies a TRUE or FALSE test field and a target field in the same business component. When the TRUE/FALSE field is true, the target field becomes read-only.

  NOTE: FieldName syntax works if FieldName is not a join field. If FieldName is a join field to another table, then this syntax does not prepopulate the field that uses this syntax in its Pre Default Value.

- **Parent Read Only Field**

  Specifies a TRUE or FALSE test business component or field combination in the parent chain (parent, grandparent, and so on) that, when TRUE, causes the target business component to become read-only.

For more information about user properties, see *Siebel Developer's Reference*.

When configuring read-only behavior, consider the following:

- Wherever a business component or field name is specified, whether in the Name or Value property of the user property object definition, make sure that the capitalization, spelling, and use of blank spaces are correct. Also make sure that quotation marks are not present.
- These user properties do not function when used in an applet in a view where the view's Admin Mode property is set to TRUE.

  Admin Mode, when TRUE, turns off all insert and update restrictions for the business components used by the view, including those specified by business component user properties. The business component Sales Rep and Personal visibility modes are ignored. Records that do not have a primary team member are also visible. However, pop-up visibility is not overridden.

  NOTE: The Admin Mode flag should be set to TRUE only in a view that is part of a screen containing all administration views. Do not use the Admin Mode flag for a view in a screen that contains any non-administration views.

  You can have a list view with Admin Mode set to TRUE that drills down to a detail view not marked as an administration view, while remaining in Admin Mode. This allows you to share detail views with non-administration list views.

BC Read-Only Field

This user property specifies a Boolean field that, when TRUE, causes all fields in the current record to become read-only. This also prevents the user from updating or deleting the record, but does not prevent the addition of new records to the business component. The Name and Value properties in the user property record are specified as follows:

- **Name**. Contains the literal text BC Read Only Field.
- **Value**. Contains the name of a field in the same business component as the parent object definition of the user property. This field must be a TRUE or FALSE field.

An example of the use of BC Read Only Field is the situation in which you need to prevent users from updating inactive accounts. The Inactive Account field in an account record is a TRUE or FALSE field that, when TRUE, indicates that the account is inactive. To configure dynamic read-only behavior for the Account business component based on this field, add a business component user property child object definition to the Account business component, with the following property settings:

- **Name**. BC Read-Only Field.
- **Value.** Inactive Account.

Field Read Only Field

This user property is similar to BC Read Only Field, in that it tests the field specified in the Value property and enforces a read-only restriction when the test field has a value of TRUE in the current record. However, unlike BC Read Only Field, the Field Read Only Field user property restricts one field in the same business component, rather than the entire business component record.

The Name and Value properties in the user property record are specified as follows:

- **Name**. Contains an expression in the following format:

  `Field Read Only Field: fieldname`

  For example:

  `Field Read Only Field: Account Status`

  Note that there is only a single space between the colon and the field name.

- **Value**. Contains the name of the test field. This is a TRUE or FALSE field in the same business component as the parent object definition of the user property.

One Field Read Only Field user property must be created for each field you want to make conditionally read-only.

An example of the use of Field Read Only Field is the situation in which you want to make the Competitor field in an account record read-only when the Type field has a value of Competitor. In other words, if an account record has been included because that account is a competitor, you do not want users to be specifying that account's competitors. The following procedure describes how to accomplish this.

To restrict the Competitor field in an account based on the account's type

1. Navigate to the Business Component object type in the Object Explorer, and then to the Account object definition in the Object List Editor.
2. Create a calculated Boolean field in this business component that will have a value of TRUE when the Type field has a value of Competitor.

   For purposes of the example, the name of this test field can be Competitor Calc, although the name is unimportant as long as it is referenced correctly in the user property.

3. In the calculation property of the Competitor Calc field, enter the following value:

   ```
   IIf([Type] = "Competitor", "Y", "N")
   ```

4 Expand the Business Component object type in the Object Explorer, and select the Business Component User Prop object type. Click the Object List Editor to make it active, and choose Edit > New Record.

5 Set the following values in the new Business Component User Prop object definition:

- **Name**. Field Read Only Field: Competitor
- **Value.** Competitor Calc

Parent Read Only Field

This property, like BC Read Only Field, places a read-only restriction on an entire business component, rather than a single target field. This restriction occurs when a TRUE or FALSE test field has a TRUE value. However, unlike BC Read Only Field and Field Read Only Field, this user property is used to place a restriction on a child or grandchild (and so on) business component of the business component containing the test field. In the other user properties, the read-only restriction is placed on the business component containing the test field, or on another field in the same business component.

Parent Read Only Field is used primarily to restrict the detail records in a multi-value group. It could also be used to restrict the detail records in a master-detail view, but in that case you need to make sure that the restricted business component is not also used in the context of some other business object than the intended one.

The Name and Value properties in the user property record are specified as follows:

- **Name.** Contains the literal text Parent Read-Only Field.
- **Value.** Contains an expression in the following format:

  buscompname.*fieldname*

  where *fieldname* is the name of the test field, that is, the TRUE or FALSE field to be evaluated, and *buscompname* is the name of the business component in which the test field is located. For example:

  ```
  Account.Inactive Account
  ```

The business component to be conditionally restricted is the one to which you add the user property as a child object definition. The business component containing the test field must be a parent or grandparent of the restricted business component by way of a link or series of link relationships.

An example of the use of this user property is the situation where you want to disable the update of the Account Address multi-value group when the account record has a Type of Competitor. To accomplish this, you add the same calculated field as in the Field Read Only Field example, and then add a user property to the Business Address business component with the following values:

- **Name.** Parent Read Only Field
- **Value.** Account.Competitor Calc

This causes the Account Address multi-value group to be read-only when the account record is for a competitor.

NOTE: When using the Parent Read Only Field user property, the test field must have its Link Specification property set to TRUE. Otherwise the dynamic read-only functionality does not work. However, if the child record is displayed in the multi-value field in the parent business component, it is not necessary to have the Link Specification property of the field set to TRUE.

Configuring Dual Currency

You can configure fields to display in multiple currencies. For example, suppose you have a global deployment in the US and in Japan and you have sales representatives in Japan who need to have the currency value displayed in the local currency. You would need to configure a field to display in both currencies.

To configure a field for dual currency

1. Create a new field in the business component to hold the currency code to which the conversion should be performed.
   - This field is not a foreign key to another table; it must be of type DTYPE_TEXT.
   - Specify PickList = PickList Currency.
   - In the corresponding Pick Map, associate Pick List Field = Currency Code with the newly created currency code field.
2. Create a new field in the business component to hold the converted currency amount.

   NOTE: You cannot configure Forecast business components to display dual currency because the list columns displaying monetary values do not map to fields. The list columns display values that are computed by buttons using specialized methods.

   - This field must be of type DTYPE_CURRENCY.
   - It must be a calculated field, Calculated Value = [Unconverted Amount]. The field Unconverted Amount must also be of type DTYPE_CURRENCY.
   - The Exchange Date Field property must point to a field of type DTYPE_DATETIME.
   - The Currency Code Field property points to the currency code field of Step 1.
3. Set the Runtime property to be TRUE in the applet that displays the converted currency.

   A pick or detail applet need not be specified, because your Siebel application automatically launches the default applet that matches the field type.
4. Before that currency conversion takes place, the underlying currency business component must be filled with a minimum number of valid values. To access the lists of currencies, conversion dates, and exchange rates in your Siebel application, navigate to Site Map > Application Administration > Currencies.
   - The two currencies between which you want to convert must be marked as active (for example, name the original one O and the converted one C).

- At least one exchange rate value must be defined for O to C. (For the reverse conversion C to O, another exchange rate value is required.)
- At least one of the exchange rates of a certain exchange direction must have a date at or before the date that is used as 'Exchange Date'.

To configure dual currency display (an example)

1 Add a field to the Opportunity business component for the currency code to which the conversion is made, as shown in the following table:

Property	Value
Name	My_Currency
Type	DTYPE_TEXT
Join	S_OPTY_X
Column	ATTRIB_03
PickList	PickList Currency

The field is stored in an unused column in the extension table S_OPTY_X.

2 Add a record to the field's child Pick Map object, as shown in the following table:

Property	Value
Field	My_Currency
Pick List Field	Currency Code

3 Add a field to the Opportunity business component for the converted revenue, as shown in the following table:

Property	Value
Name	My_Cvt_Revenue
Calculated	TRUE
Calculated Value	[Revenue]
Currency Code Field	My_Currency
Exchange Date Field	Sales Stage Date
Type	DTYPE_CURRENCY

4 Add two new list columns to the Opportunity List Applet, as shown in the following tables:

Property	Value
Field	My_Currency
Display Name	Converted Currency Code

Property	Value
Field	My_Cvt_Revenue
Display Name	Converted Revenue
Runtime	TRUE

5 Compile the Oppty and Oppty (SSE) projects.

Exposing Business Components as OLEDB Tables

OLEDB is a specification for a set of data access interfaces designed to enable heterogeneous data stores to work together. Components built to the OLEDB standard behave as a table, even though complex computing processes can occur between the data sources and the applications.

The Siebel OLEDB Rowset wizard is a read-only provider that exposes Siebel business components as virtual OLEDB tables. Using the Siebel OLEDB Provider, external OLEDB-enabled applications can access data stored in Siebel by referring to Siebel objects like Contact or Account without the need to understand the internal functioning of the Siebel Data Model. You can configure the Siebel business components that are exposed to the client application as OLEDB tables.

The OLEDB Rowset wizard steps you through the process of creating OLEDB tables.

For more information about the OLEDB Rowset wizard, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*.

Configuring Client-Side Import

You can use client-side import to populate fields with data. Client-side import uses the applet-level menu import functionality in the Siebel Web Client. It is configured using the Import Object object in Siebel Tools, which sets business component fields to be populated. Client-side import is supported for parent business components only. Applets configured for client-side import must allow records to be inserted; that is, the *No Insert* property of the applet must be set to *False*.

For an example of how client-side import is configured, see the Contact business component in the standard Siebel repository. The contact business component is configured as an Import Object with fields defined as Import Field child objects.

To enable client-side import for a business component

1. Lock the project to which the business component belongs.
2. In the Object Explorer, select Import Object.
3. Add a new record with the business component and its project as properties.
4. With the new record selected, expand Import Object, and then select the Import Field child object.
5. Add new records for each business component field you wish to be populated.

   NOTE: You can also add import fields to already existing import objects, such as Contact.
6. Compile the .srf, selecting the locked project.

The new fields will be displayed in the Select a Siebel Field dialog and can be mapped to fields in the External Data Source Field dialog when importing data.

Managing Unused Business Components

Generally, any supplied unused objects must remain intact and must not be deleted, inactivated, or renamed. Do not delete these definitions because other objects may reference them. Delete any custom business components that are not being used and do not reference any other object definition, such as an applet.

8 Configuring Joins

Topics in This Chapter

About Joins

A Join object definition creates a relationship between a business component and a table other than its base table. This relationship allows the business component to use columns from the other table. The join uses a foreign key in the business component to obtain rows on a one-to-one basis from the joined table, even though the two tables do not necessarily have a one-to-one relationship.

Figure 36 shows the Service Request List Applet that exposes the Account field. A join brings the Account field into the Service Request business component and the Service Request List Applet displays the data in the user interface.

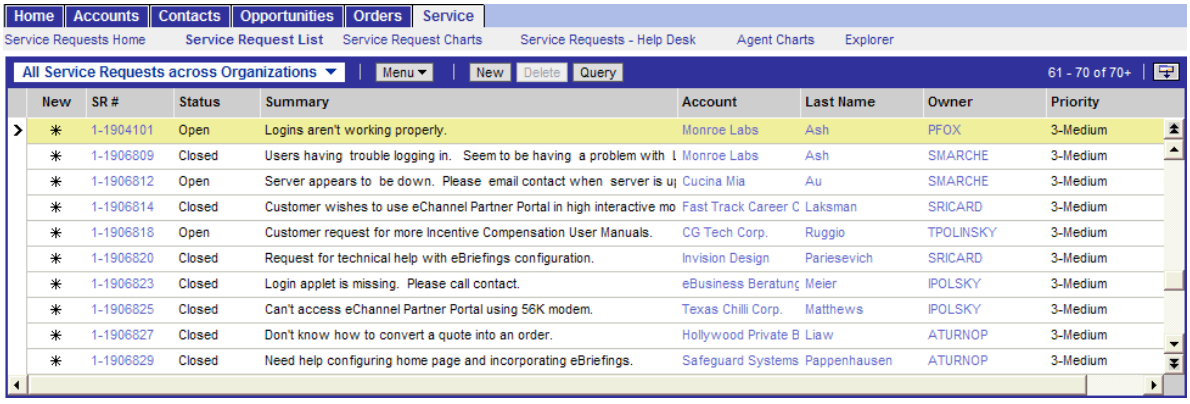


Figure 36. Example of Joined Field Exposed in the User Interface

The master-detail relationship is implemented with a foreign key column in the detail table (as shown in Figure 37). Multiple rows in the detail table have the same foreign key value pointing back to the same row in the master table. A join is always one-to-one and it is always between a business component and a table. After a join is created, you can create additional fields in the business component based on columns in the joined table.

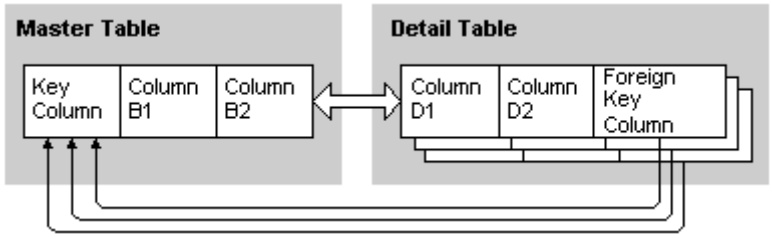


Figure 37. Master-Detail Relationship in a Join

NOTE: It is possible to use a joined field as Source Field on the join specification. This is important if, for example, you need to join in grandparent data through the parent id field on the parent business component.

About Implicit Joins

Implicit joins exist as part of the Siebel object architecture. They are not explicitly defined using Siebel Tools. Unlike joins that you define in Siebel Tools, users can update the columns from implicit joins.

Implicit joins exist for the following:

- All 1:1 (_X) extension tables and all relevant intersection tables.
- Extension tables of S_PARTY, such as S_ORG_EXT, S_CONTACT, S_POSTN, and S_USER. These implicit joins are used to map data to party business components. For example, if you were to add a field to the Account business component and then selected the Join property, you would see several implicit joins that do not appear in the Join list displayed in the Tools Object List Editor, including joins with an alias of S_ORG_EXT and S_USERs.

Implicit joins usually use the table name as the Join Alias.

For more information about implicit joins, see "About Extension Tables" on page 92.

How a Join Is Constructed

A join is constructed using the objects and relationships shown in Figure 38.

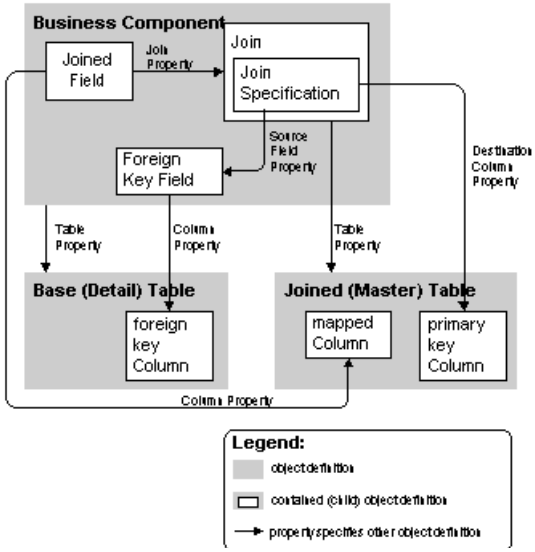


Figure 38. Join Architecture

The roles of the object in the diagram are summarized as follows:

- **Business Component object type.** The business component is the parent object definition of the join. Because of the join, fields in the business component (called joined fields) can represent columns from the joined table.
- **Joined field.** A joined field in the business component represents a column from a table other than the business component's base table. Therefore, a joined field must obtain its values through a join. A joined field has the name of the join in its Join property. Together the Join property and Column property identify the column and how to access it. When creating a joined field in a business component, you can change the Type property from the default DTYPE_TEXT to a more appropriate type. For example, if you are joining a table column that contains phone numbers, you can change the Type field to DTYPE_PHONE.
- **Join object type.** Join is a child object type of the Business Component object type. The Join object definition uniquely identifies a join relationship for the parent business component and provides the name of the destination (joined) table. The join object definition identifies the joined table in the Table property. The name of the base table is already known to the business component.

  NOTE: Set the Outer Join Flag to TRUE if you want to retrieve all the records in the business component even when the joined fields are empty.

- **Join Specification object type.** The join specification object definition is a child of the join object definition. It identifies the foreign key field in the business component and the primary key column in the joined table (that the foreign key points to).

  The Source Field property identifies the foreign key field in the business component. If left blank, the Source Field is the Id field, indicating a one-to-one relationship between the business component and the joined table. Occasionally, a system field such as Created By or Updated By may be specified as the foreign key field in the Source Field property.

  The Destination Column property identifies the primary key column in the joined table. A nonblank Destination Column property value is required if the join occurs on a column other than ROW_ID. A blank value in the Destination Column property means that the destination column is ROW_ID, which is typically the primary key in tables in Siebel applications.

  NOTE: In rare circumstances, there can be multiple join specifications in a single join. For example, the Sub Campaign business component has a join to the S_LANG table with two join specifications. In such cases the source fields in the join specifications should be based upon the same table.

- **Join Constraints.** A join constraint is a constant-valued search specification applied to a column during a join. It is for use with outer joins.

- **Foreign key (source) field and foreign key column.** The foreign key field is identified in the Source Field property of the join specification. It represents a foreign key column in the base table, pointing to rows in a particular table used in joins. For example, in the Contact business component, the foreign key field to the join on accounts data is the Account Id field, which represents the PR_DEPT_OU_ID column in the base table.

- **Joined table.** The joined table is the master table in the master-detail relationship. It provides columns to the business component through the join. The joined table is identified in the Table property of the Join object definition.

  NOTE: When configuring a recursive or self join, the Alias name of the joins must be different than the Table Name. Using the same name will result in the following error message: "Table 'T1' requires a unique correlation name." This error is often due to a faulty recursive or self join definition.

- **Primary key (destination) column.** The join specification identifies the primary key column in the joined table (in the Destination Column property). Every standard table in standard Siebel applications has a ROW_ID column that uniquely identifies rows in the table. ROW_ID is the destination in most joins.

- **Mapped column.** Columns in the joined table are available for use in fields in the business component.

Guidelines for Configuring Joins

A join defines a logical relationship between the base table of a business component and another table. The join is a child object of a business component. Fields in a business component reference joins. A join should only be used when the resulting database join will retrieve no records (zero) or only one record. For example, a join is used to retrieve the primary Account for an Opportunity.

When configuring joins, consider the following:

- A business component may have more than one join with the same destination table if you specify an alias for each join using the Alias property. For example, the Action business component may have two joins to the S_CONTACT table, one to retrieve the owner of the person who created the activity, and another to retrieve the contact associated to the activity. In this example, the joins aliases are Owner and Primary Contact respectively.
- It is important that the Alias property of the join be distinct even though the destination table is the same. It is usually not a good practice to use the table name as the Alias name, even though this is common in the standard repository. This is because implicit joins will use the table name as the Alias to make sure that the explicit join name is not used instead. To make sure that no conflict exists, you should always give the join a distinct and custom alias name.
- For joins to nonparty related tables, the only column you can update is the field (in the parent business component) with the foreign key value. You must specify the table to join to and whether it is an outer join. You must also specify the join specification definition with the source field in the parent business component that stores the foreign key value and the destination column in the child table, which is usually ROW_ID.
- For joins to party-related tables, those that are extensions of the S_PARTY table (such as S_CONTACT, S_ORG_EXT, S_USER, or S_POSTN), require that the foreign key value be exposed as the source field. However, unlike joins to nonparty related tables, the destination Column must reference the PAR_ROW_ID column in the joined table.

Typical scenarios for creating new joins are:

- When a join to a particular table does not already exist within the business component definition and there is a foreign key value between the table the business component is based on and the joined table.
- When the foreign key value is stored in a field that is not already defined as a source in an existing join.
- When bringing in party data into a non-party business component, create a new join with the join specification based on PAR_ROW_ID.
- When bringing in party data into a party business component, use the appropriate explicit join.
- When mapping fields in party business components, use the implicit join for the extension table.

Using a Predefault Value for a Join Field

Since a join field cannot be updated, you cannot use a predefault value in the regular way as a default field value if nothing is specified when a record is inserted. You can use a predefault value for a join field to show the join field value immediately as the new record is being inserted.

To use a predefault value for a join field

The following procedure uses the Opportunity Product business component as an example.

1. Define a join to S_OPTY in the Opportunity Product business component.

2. Define two new fields based on the join to show Opportunity Sales Stage and Name.
3. Add the two fields to the Opportunity Product applet.
4. Compile and test using the standard Opportunities—Products view.
   a. Add a new Product for an Opportunity.
   b. Note that the join fields are not populated until you requery the applet.

      (However, the source field—Oppty Id—for the join is populated.)
5. Set the Predefault properties of the new fields to Parent: 'ParentBusinessComponent.JoinedField'.

   For example, predefault Opportunity Name with Parent: 'Opportunity.Name' and predefault Opportunity Sales Stage with Parent: 'Opportunity.Sales Stage'.
6. Set the Link Specification property of the joined fields (Name and Sales Stage) in the parent business component to TRUE.
7. Compile and then add a new product for an Opportunity.

The join fields are populated immediately, and you do not need to requery the applet to see them.

9 Configuring Links

Topics in This Chapter

About Links

A link defines a one-to-many (or master-detail) relationship between two business components. The Link object type makes master-detail views possible, in which one record of the master business component displays with many detail business component records that correspond to the master. A master-detail relationship appears in Figure 39, showing an account in the form and the many contacts for the account.

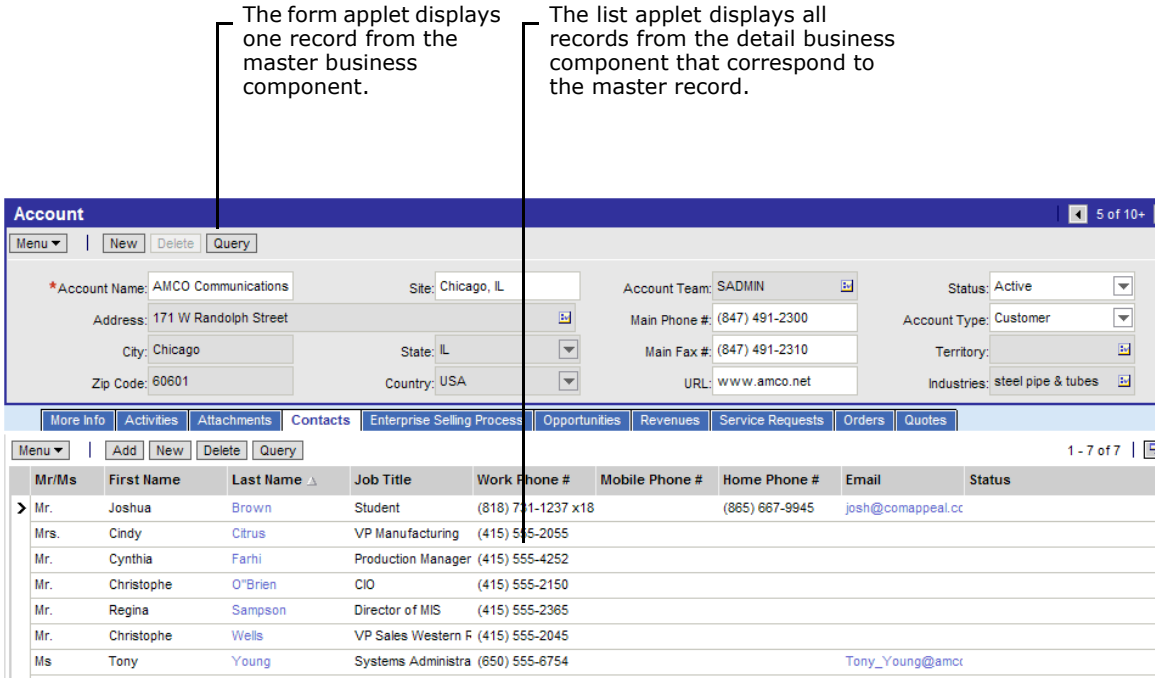


Figure 39. Link in a Master-Detail View

In this master-detail view, each account record can be associated with many contact records. The synchronization between the master and detail business components in a master-detail view is accomplished with a link between the two business components and the inclusion of the link and business components in a business object. Business objects are described in "About Business Objects" on page 223.

NOTE: Link destination fields are initialized automatically when you add a record to the child business component in a link.

Links are also used in the implementation of multi-value group applets. A multi-value group applet is a dialog box that displays multiple records of data associated with one control in the originating applet. For more information, see "About Multi-Value Links" on page 211.

How Links Are Constructed

The relationships between object definitions used to implement a link appear in Figure 40.

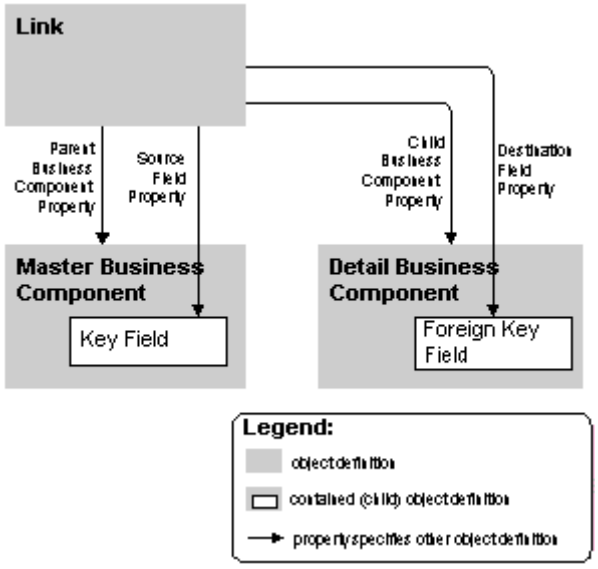


Figure 40. Link Property Relationships

The object definitions in Figure 40 are as follows:

- **Link.** The Link object definition specifies a master-detail relationship between two business components. It identifies the master and detail business components, the key field in the master business component, and the foreign key field in the detail business component.
- **Master business component.** The master business component is the "one" in the one-to-many relationship. The name of this object definition is specified in the Parent Business Component property in the Link object definition.
- **Detail business component.** The detail business component is the "many" in the one-to-many relationship. The name of this object definition is specified in the Child Business Component property in the Link object definition.

  NOTE: The Calendar business component should not be used as the master or detail business component in a link.
- **Source (primary key) field.** The source field, also known as the primary key field, is a field in the master business component that uniquely identifies records in the business component. It represents the ROW_ID column from the business component's base table. The name of this field is specified in the Source Field property in the Link object definition. Source field typically, but not necessarily, represents the row id column from the business component's base table.

- **Destination (foreign key) field.** The destination field, also known as the foreign key field, is a field in the detail business component that points back to the master record in the business component. Account Id and Opportunity Id are typical foreign key fields. A foreign key field represents a foreign key column from the detail business component's base table, such as PR_DEPT_OU_ID (the base table for the Account business component). The name of this field is specified in the Destination Field property in the Link object definition.

  In a link based on an intersection table, that is, one in which the Inter Table, Inter Parent Column, and Inter Child Column properties are non-blank, you do not specify the Source Field or Destination Field properties. Otherwise, the Destination Field property needs to contain the name of a field in the base table of the business component (not based on a join), and the field has to be updated.

  NOTE: For a M:M link, you could specify a source field. Destination will always default to Id, even if another value is specified.

A link in a master-detail view is implemented using the object types illustrated in Figure 41.

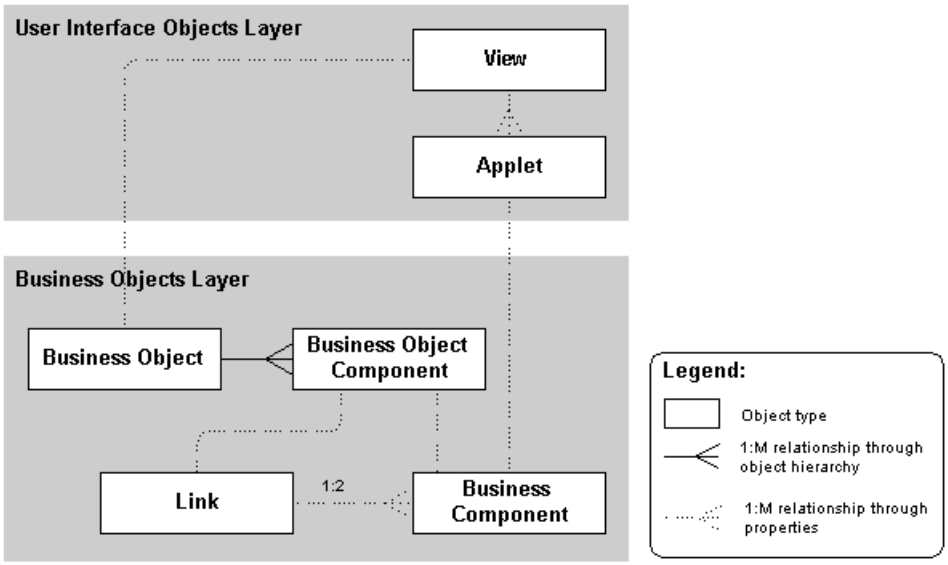


Figure 41. Link Architecture

In a master-detail view, a Link object definition is incorporated into a business object (by means of a Business Object Component object definition) to establish the master-detail relationship. This relationship applies to any use of the two business components together within the context of the business object. Each view specifies the business object it uses in its Business Object property. This forces the view to operate as a master-detail view, as specified in the link, without any additional configuration of the view. For more information about master-detail views, see "About Views" on page 281.

Creating Links to Define a 1:M Relationship

In a one–to–many link, you define the child business component and the destination field (the foreign key to the parent) of that business component. You also define the parent business component and the source field of that business component. If the source field is not defined, it defaults to the parent business component's ID. An example of this type of link is the Account/ Account Note link. One Account can have many note records. The ID (source field) of the parent Account record is stored in the Account Id (destination field) field of the Note record.

Using Two Links to Define a M:M Relationships

Two link object definitions with opposite master-detail settings are used to establish a many-to-many relationship based on an intersection table. The Inter Table, Inter Parent Column, and Inter Child Column properties of the two Link object definitions are used to establish the connection between the links and the intersection table. An example of this type of link is the Opportunity/Account. The intersection table is S_OPTY_ORG. The Inter Child Column is OU_ID, which is the ID in the Account business component. The Inter Parent Column is OPTY_ID, which is the ID in the Opportunity business component.

For information on the use of M:M links, see "About Intersection Tables" on page 97.

About the Cascade Delete Property

The Cascade Delete property determines whether a child record is deleted when the parent record is deleted. Take special precautions when determining this property. If set incorrectly, it may cause data integrity issues or orphaned records.

The Cascade Delete property can be set to the following values:

- **Delete.** The detail records are deleted along with the master. This is most appropriate for values stored on _XM tables where the only related record is the parent record.

  Do not use DELETE if the child business component in this link is also a detail business component in another link. In this case, use CLEAR instead.

- **Clear.** The foreign key reference is removed if the master record is deleted, but the detail records remain in place. However, the value in the foreign key column is cleared. This is most appropriate for child values that might be shared with other parent records.

- **None.** No records are deleted and the foreign key column is not cleared. This is the default setting.

Cascade Delete is not available for many-to-many links. With a many-to-many link, Siebel applications will automatically delete the intersection record but will leave the child record intact, as it may have other parents.

When you delete a record which is pointed to by foreign keys of other tables, the references to it may or may not be deleted. If those references are not deleted, the user is left with row IDs which point to nonexistent records. In the case of multi-value groups, sometimes these foreign keys will be converted to say "No Match Row Id."

About the Search Specification Property

The Search Specification property of a link is applied to the child business component. Be aware that a Search Specification can also be applied at the applet level, and any Search Specifications that exist at the applet level will be added to this Search Specification by using the query operator AND.

NOTE: Sort Specification applies only to association lists.

About the Visibility Rule Property

The Visibility Rule property of link determines whether visibility will be applied in the Link. Possible values are Never and Always.

- **Always.** Allows visibility rules in the detail records when the current master-detail view is based on this link, even though the view does not have active visibility settings in the Visibility Applet and Visibility Applet Type properties.
- **Never.** Disables visibility rules in the detail records when the current view is based on this link.

Using the Check No Match Property with a Primary Join

When a multi-value link has been configured with a primary join—which is the typical situation—there are circumstances in which the foreign key used by this join to identify the primary record is unable to find the primary. For example, this can happen when the primary record has been deleted from the multi-value group or the multi-value group is newly created and has no records. In such cases, the multi-value link can be configured to update the primary foreign key to a value of NULL, or to a special value of NoMatchRowId, depending on your requirements. This behavior is configured through the Check No Match property of the Multi Value Link object type, and has performance consequences.

The purpose of the special NoMatchRowId value is to prevent secondary queries on foreign key values that are known to have failed, thereby improving performance, much in the same way that using a primary join improves performance.

The NoMatchRowId generating and testing behavior is activated by setting Check No Match to FALSE for the MVL. This setting has the following results:

- When the application encounters a master record where the primary foreign key is NULL or invalid, it performs a secondary query to determine if there are detail records in the multi-value group. If it finds there are no detail records, it sets the primary ID field to the special value NoMatchRowId.

- When the application encounters a master record where the primary foreign key has the value "NoMatchRowId," this indicates to the system that there are no detail records in the multi-value group and the secondary query is not performed.

If you set Check No Match to TRUE, the Siebel application will perform a secondary query whenever the outer join on the primary fails, or is set to NULL or NoMatchRowId. If the secondary query finds a matching detail record, it updates the foreign key with that record's row ID, provided the MVL has an Auto Primary property setting of DEFAULT. If no matching child record is found, or Auto Primary is set to NONE, the application leaves the existing value intact.

A Check No Match setting of TRUE can have serious negative performance consequences. If a multi-value group is sparsely populated (that is, most master records do not have any detail records in the multi-value group) and has Check No Match set to TRUE, it will be almost as slow as not having a primary join at all.

Check No Match should be set to FALSE for most multi-value links because of the performance consequences. It should only be set to TRUE if the multi-value group could possibly have records added to it without going through the MVG itself. For example, account addresses might actually be inserted by means of the Business Address multi-value group on the Contact business component instead of the Address multi-value group on the Account business component. Also, if records can be added to the detail business component through EIM, the TRUE setting is the appropriate one.

The Use Primary Join property should be set to TRUE if CheckNoMatch is TRUE. If CheckNoMatch is set to TRUE and Use Primary Join is FALSE, then the Siebel application will always do the secondary query to find the child records.

About Multi-Value Links

The Multi Value Link object type is a child object type of the Business Component object type. A *multi-value link* implements an *multi-value group*, a list of records attached to a control or list column in an applet.

The relationship between the business component of the originating applet and the business component of the multi-value group applet is one-to-many; that is, a master-detail relationship. This master-detail relationship, as with all master-detail relationships in Siebel applications, is implemented through a Link object (in addition to other object types). However, a multi-value link is also necessary to adapt a link for multi-value group applet use.

NOTE: The relationship between the two business components is one-to-many in the context of the multi-value link and multi-value group. There may be, in fact, a many-to-many relationship (for example, between opportunities and positions), but in the context of the multi-value group, only one master-detail relationship is presented.

An example of a a multi-value group applet is shown in Figure 42.

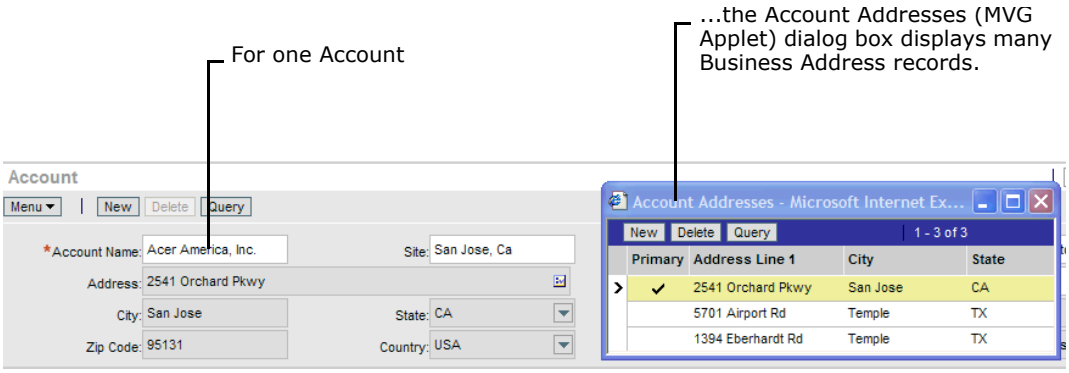


Figure 42. Multi-Value Group Applet Example

An account can have multiple addresses. These are stored in the Business Address business component. Clicking the Select button to the right of the Address field opens the Account Address dialog box. This dialog box lists the addresses, including the street address, city, state, and ZIP Code associated with each account. It also allows a user to add, delete, or modify individual records.

In the Account Form Applet, the Address, City, State, ZIP and Country text boxes display the values from the corresponding fields in the primary record in the Business Address business component. The primary record is indicated in the multi-value group applet with a check mark in the list column labeled Primary. You can select a different primary record by clicking the Primary list column in a different record.

The fields in the master business component (Account in the illustration) that are populated by the primary record in the multi-value group business component are called *multi-value fields.*

NOTE: If you want to query the originating applet for all master records that have a detail record with a specific field value you can only use multi-value fields.

Multi-value fields are populated with data from a record in the detail business component because of the multi-value link. Multi Value Link is a child object type of Business Component that defines a master-detail relationship (based on a link) to embed in the business component. These embedded master-detail relationships are used to expose fields from the detail business component as fields directly in the master business component.

NOTE: Most, but not all, multi-value links are set up to designate a primary record. Those that do not designate a primary use the first record retrieved from the detail business component. For more information, see "Configuring the Primary ID Field" on page 219.

How Multi-Value Links Are Constructed

A multi-value link is based on a link object definition; the link is referenced in the Destination Link property of the Multi Value Link object definition. It is the link object definition that specifies the one-to-many relationship between the master and detail business components. The multi-value link object definition performs two roles:

- To give fields in the master business component access to primary record field values through the link.
- To allow embedding of detail data in the same business component as master data, so both can appear in the same applet.

The object types illustrated in Figure 43 participate in the configuration of a multi-value link.

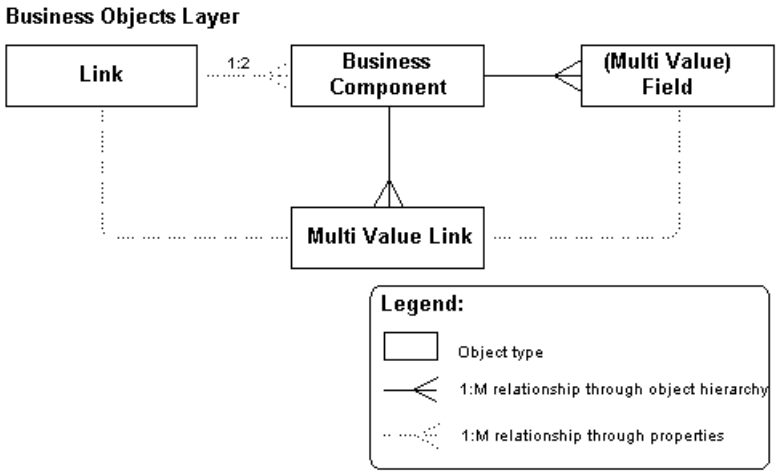


Figure 43. Multi-Value Link Architecture

The object type box in the diagram labeled (Multi Value) Field indicates that either field or multi-value field is correct for referring to this object type in this context. Multi Value Field is a distinct object type, but only in the sense that it can be accessed in the Object Explorer. It is only a representation of the Field object type. Multi-value fields are those fields that have a nonblank Multi Value Link property and a Multi Valued property value of TRUE; all other fields are single-value fields.

The details of the object definition relationships appear in Figure 44.

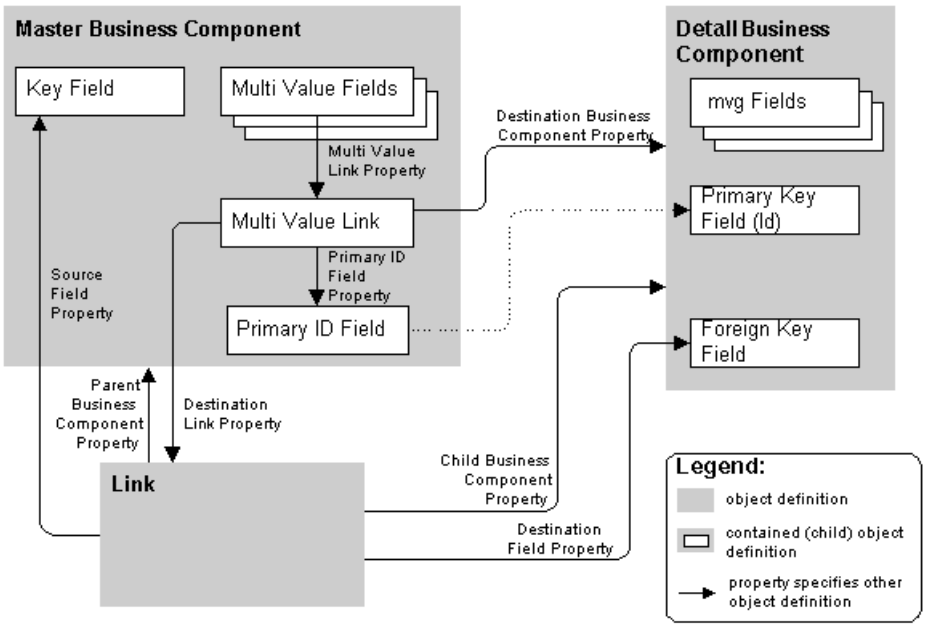


Figure 44. Multi-Value Link Details

The roles of the object definitions in Figure 44 are explained in the following list. References to the address example refer to the Account Addresses dialog box shown in Figure 42.

- **Master business component.** The master business component is the "master" in the master-detail relationship specified in the link. Fields from this business component are displayed in the applet from which the multi-value group applet is initiated. The master business component in the Account Addresses dialog box example is Account.
- **Multi-value fields.** Multi-value fields are fields in the master business component that are populated by the current (typically primary) record in the detail business component through the multi-value link and link object definitions. Each of these fields has the name of the multi-value link specified in its Multi Value Link property, and a Multi Valued property setting of TRUE. A multi-value field has a blank Column property setting because its values are obtained from the current record in the detail business component, rather than from the master business component's base table.
- **Key field.** The key field in the master business component is the primary key for that business component. The key field is referenced in the Source Field property of the Link object definition.
- **Multi Value Link object.** The Multi Value Link object definition defines the relationship between the link object definition and fields in the master business component, using the following properties:
  - **Destination Link.** Identifies the link.
  - **Destination Business Component.** Identifies the detail business component.

- **Primary Id Field.** Identifies the field in the detail business component that designates which record is the primary.

In the example, the multi-value link is called Business Address.

- **Link.** The Link object definition specifies a master-detail relationship between the two business components. The Link object definition can be used in other contexts, such as master-detail views or other multi-value links. The multi-value link identifies the link in its Destination Link property. In the address example, the link is Account and Business Address.
- **Detail business component.** The detail business component supplies the detail records in the master-detail relationship. In the address example, this is Business Address.
- **Foreign key field.** The foreign key field contains row ID values that point back to records in the master business component and uniquely identify the master for each detail business component record. The foreign key field is used in the specification of the link; the link identifies the foreign key field in its Destination Field property. In the address example, the foreign key field is Account Id.

  NOTE: There is no foreign key field specified in a link based on an intersection table.

- **Primary ID field.** The primary ID field in the master business component holds the row ID value of the primary record for each multi-value group in the detail business component. It is identified in the Primary Id Field property of the multi-value link. The primary ID field allows the primary detail record to be identified for each master record. For more information, refer to "Configuring the Primary ID Field" on page 219.

How Indirect Multi-Value Links Are Constructed

An indirect multi-value link has a parent business component that related to the business component on which the MVL is defined by a join. The source field of an indirect multi-value link is based on a column that is joined in from another table and not a column in the base table. The configuration of an indirect multi-value link is illustrated in Figure 45.

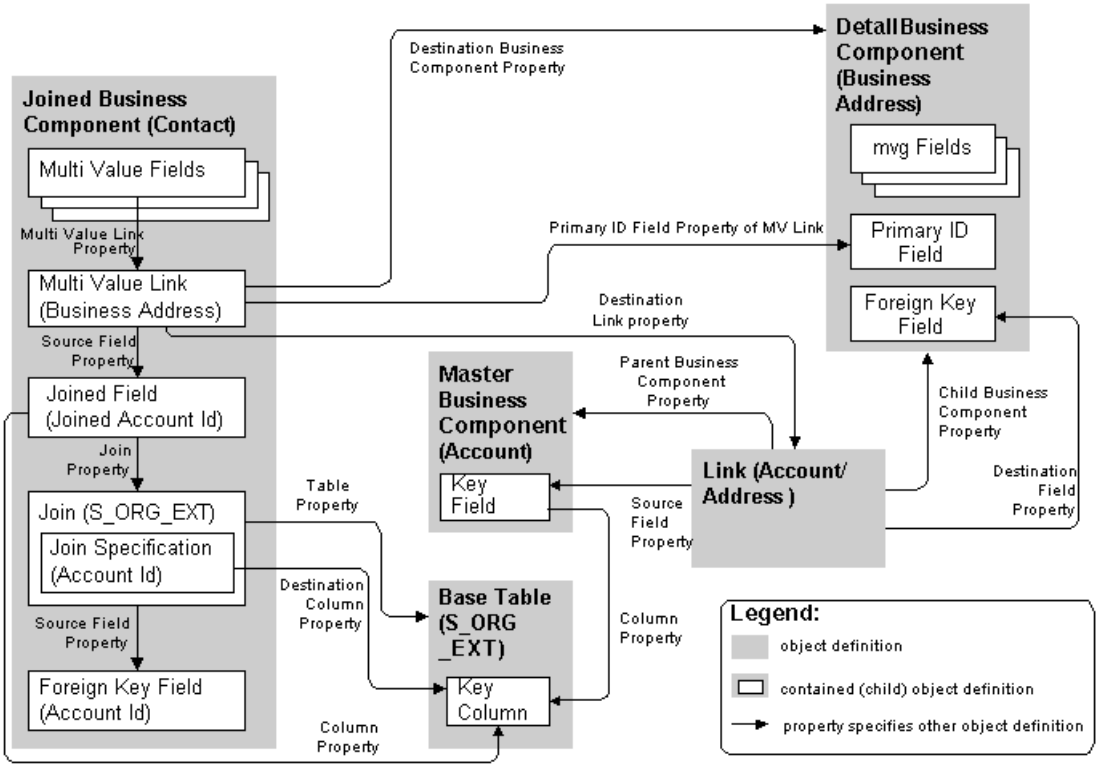


Figure 45. Indirect Multi-Value Link Details

In Figure 45, the object definition names have been provided for an example multi-value link called Business Address in the Contact business component. Although given the same name as its counterpart in the Account business component, this is a different object definition. For a comparison of conventional and indirect multi-value links, review Figure 44 on page 214.

The primary difference between the Business Address multi-value link in the Contact business component and in its Account counterpart is that the multi-value link object definition is found in a business component other than the master business component.

The Source Field property in the multi-value link in the Contact business component is nonblank. In a conventional multi-value link this property is blank, indicating that the Id field in the current business component is used (corresponding to ROW_ID in the base table). In the indirect multi-value link, the Source Field property specifies a field in the S_ORG_EXT join called Joined Account ID. The Joined Account ID field provides the Account Id of the Account that corresponds to the current Contact.

The roles of the object definitions in Figure 45 on page 216 are explained as follows:

- **Join business component.** The join business component has a master-detail relationship with the master business component in the link. In this relationship, the join business component is the detail rather than master. The indirect multi-value link is established as a child object definition of the join business component.
- **Multi-value fields.** Multi-value fields are fields in the join business component that are populated by the primary record in the detail business component through the multi-value link and link object definitions. Each field has the name of the multi-value link specified in its Multi Value Link property, a Multi-Valued property setting of TRUE, and a blank Column property.
- **Multi-value link.** The Multi-Value Link object definition defines the relationship between the link object definition and fields in the master business component, using the following properties:
  - **Destination Link.** Identifies the link.
  - **Destination Business Component.** Identifies the detail business component.
  - **Primary Id Field.** Identifies the field from the business component that the MVL belongs to.
- **Joined field.** In a conventional multi-value link, the Source Field property is blank. In an indirect multi-value link, the Source Field property specifies a joined field in the same business component as the multi-value link. The joined field represents the ROW_ID column from the base table of the master business component. The ROW_ID column is obtained by means of a join.
- **Join and join specification.** The Join and Join Specification object definitions make it possible to populate the joined field.
- **Foreign key field (in the joined business component).** The foreign key field represents a foreign key column in the base table. The foreign key field points to rows in the joined table, in this case the base table of the master business component. The foreign key field is used in the implementation of the join.
- **Master business component.** The master business component is the "master" in the master-detail relationship specified in the link. The master business component in the example in Figure 45 on page 216 is Account.
- **Base table.** The join, join specification, and foreign key field in the join business component access the base table of the master business component. This makes possible a join relationship that provides a master business component record and, indirectly, a set of detail business component records for each join business component record.
- **Key field.** The key field in the master business component is the primary key for that business component. The key field is referenced in the Source Field property of the Link object definition.
- **Link.** The Link object definition specifies a master-detail relationship between the master and detail business components.

- **Detail business component.** The detail business component supplies the detail records in the master-detail relationship.
- **Foreign key field (in the detail business component).** The foreign key field contains row ID values that point back to records in the master business component. These row ID values uniquely identify the master for each detail business component record. The foreign key field is identified in the link in the Destination Field property.
- **Primary ID field.** The primary ID field in the master business component holds the row ID value of the primary record for each multi-value group in the detail business component. The primary ID field is identified in the Primary Id Field property of the multi-value link. The primary ID field allows the primary detail record to be identified for each master record. For more information, refer to "Configuring the Primary ID Field" on page 219.

The parent component of a multi-value link is usually the same as the business component in which the MVL is defined. However, by using the Source Field property of the Link object, it is also possible to create an MVL whose parent business component is related to the current business component indirectly using a join or another MVL.

Every MVL in a Siebel application is based on an underlying Link object, whose name is specified by the Destination Link property of the multi-value link. Every link, in turn, defines a one-to-many relationship between two business components. Typically, the business component in which an MVL is defined is the same as the parent business component of the underlying link on which the MVL is based.

For example, consider the Business Address multi-value link in the Account business component:

```
[MultiValueLink]

   DestBusComp = "Business Address"
   DestLink = "Account/Business Address"
   PrimaryIdField = "Primary Address Id"
   CheckNoMatch = "TRUE"
   PopupUpdOnly = "TRUE"
   NoCopy = "TRUE"
```

The Destination Link property indicates that this MVL is based on the Account/Business Address link, which is itself defined as:

```
[Link]

   Name = "Account/Business Address"
   ParentBusComp = "Account"
   ChildBusComp = "Business Address"
   DestField = "Account Id"
   CascadeDelete = "Delete"
```

The parent business component of this link is the Account business component, which is also the business component in which the MVL has been defined. In this typical MVL configuration, the multi-value group will be populated with all the children Business Address records for whichever Account is currently selected in the Account business component.

How a Cascade Copy with a Multi-Value Link Is Constructed

It is often desirable to be able to configure a business component to support the copying of its detail records when one of its records is copied. You implement this with a feature called *cascade copy*. For example, when you make a copy of an opportunity record to create a similar opportunity, you may always want the list of contacts for that opportunity copied with it.

To implement the cascade copy feature for a business component, you create a Multi Value Link child object definition and specify the following properties:

- **Destination Link.** The name of the link in which the master-detail relationship is specified.
- **Destination Business Component.** The name of the detail business component.
- **No Copy.** This must be set to FALSE and #Field ID should be set to No Copy = FALSE. If the No Copy property is set to TRUE, cascade copying is disabled. However, an exception to this occurs when the corresponding field is defined as the Destination field in a link. In this case, the link automatically populates the field and ignores the value of the No Copy property.

A multi-value link used in the implementation of a multi-value field automatically copies the detail records (unless disabled with No Copy) because it is assumed that a multi-value group travels with its parent record. For example, you would normally want the account addresses, sales team, and industry list for an account to copy with that account.

This capability is used for a different purpose when cascade copy is implemented for a multi-value link not used in a multi-value field. The multi-value link does not need to be attached to a field in the business component, or used in a multi-value group. It just needs to be created as a child object definition of the master business component, configured to point to the detail business component and link, and set with copying enabled (a No Copy value of FALSE).

Cascade copy can be implemented for a many-to-many relationship, that is, where the destination link has a nonblank Inter Table property value. In this circumstance, new intersection table rows are created rather than new detail business component records. New associations are created rather than new records. These associations are between the new master and the existing detail records.

NOTE: Cascade copy has the potential to violate the requirement of uniqueness of values in indexes. For this reason, if copying the detail records would cause any unique index violations, the copy operation is cancelled.

Configuring the Primary ID Field

The Link and Multi Value Link object definitions have a set of properties that you can use to specify to the system how to obtain the record ID of the first record to display of the detail table each time the master record changes. These properties are Primary Id Field, Use Primary Join, and Auto Primary. Together they implement the primary ID field.

The basic concept behind a primary ID is that it is faster for a Siebel application to retrieve one primary record from the MVG business component through a join than retrieve all of them through a subquery—especially because users can see values from only one child record until they open up the MVG applet. To create a primary field for a one-to-many or many-to-many relationship, complete the following procedure.

To configure a primary field for a 1:M or M:M relationship

1. Create a Primary Id column.
2. Create a field based on that Primary Id column.
3. In a Multi-Value Link, set the Primary Id Field attribute to the new Primary Id field.
4. Set the Use Primary Join attribute to TRUE.

For example, in the Account business component the primary ID field for the Address multi-value group is called Primary Address Id. The Account Address Mvg Applet displays the corresponding multi-value group. The primary record, indicated with a check mark in the list column labeled Primary, has its row ID stored in the Primary Address Id field in the account record. Each time there is a different account record displayed, the multi-value fields for the Address load the primary Business Address record's values only. It is not necessary to query the Business Address business component for multiple rows. This can be a significant performance enhancement, especially in list applets.

NOTE: In a multi-value group applet, the list column that displays the check mark (indicating the primary or nonprimary status of each record) obtains its data from a system field called SSA Primary Field. This field does not appear in the Object Explorer or Object List Editor, but may be referenced by a list column for this purpose.

The benefit of using a primary ID, from the system's standpoint, is that it converts a one-to-many relationship into a one-to-one relationship. This allows the row retrieval process to be simplified from a query with subqueries to a simple join query. This substantially improves performance, especially when the user is scrolling through the records of a list applet that displays the master.

The properties of Link or Multi-Value Link object types used to implement a primary ID field are as follows:

- **Primary ID Field.** This property specifies the name of the field in the master business component that holds the row ID values pointing to primary records in the detail business component.
- **Use Primary Join.** The Use Primary Join property is a TRUE or FALSE property that turns the Primary Join feature on or off. If TRUE, the primary detail record is obtained for each master record through a join on the primary ID field. If FALSE, the detail table is queried again with each master record change.
- **Auto Primary.** This property setting determines how row ID values are populated in the primary ID field, based on a system-supplied list column labeled Primary in the multi-value group applet. The user can manually select the primary. Auto Primary determines how, if at all, the primary selection is defaulted. The possible values for Auto Primary are DEFAULT, SELECTED, or NONE as follows:
  - **DEFAULT.** The first record automatically becomes the primary.

- **SELECTED.** The highlighted record becomes the primary when the user views the multi-value group applet and then exits.
- **NONE.** The user must manually specify the primary.

SELECTED only pertains when there are several multi-value links pointing to the same detail business component. This is the case for the Bill To Business Address and Ship To Business Address multi-value links in a standard Siebel Sales application. These multi-value links exist under both the Order and Account business components. In this case, an example of the desired behavior is as follows: if a primary is not set for the Bill To address, then when the Siebel application does a separate query to bring back all addresses associated with the account (or order), it will check to see whether one of the addresses has already been selected as primary for the Ship To address and, if so, it will SELECT (that is, set) that address as the primary for Bill To address as well.

When the Auto Primary property of a Multi Value Link object has a value of SELECTED, setting read-only properties at the applet level still does not force the SSA Primary Field to be read-only.

NOTE: If the destination business component of the Multi-Value Link is read-only, you may receive the following error message, "This operation is not available for a read-only field 'SSA Primary Field'." This is because the Primary ID field is automatically updated through the system field 'SSA Primary Field', which belongs to the destination business component. Additionally, if this business component is read-only, the field is read-only as well and cannot be updated.

Allowing Users to Set Primaries

You can set the MVG Set Primary Restricted: *visibility_mvlink_name* user property in the business component underlying the MVG applet to allow certain users to set primaries. Setting this user property to FALSE allows the Primary team member to be altered by someone other than the Manager or Siebel Administrator.

If this user property is not set, only Siebel Administrators (in Admin mode) and Managers (in Manager view mode) have the ability to change the Primary team member on opportunities, accounts and contacts.

For more information about user properties, see *Siebel Developer's Reference*.

10 Configuring Business Objects

Topics in This Chapter

About Business Objects

A business object represents a major functional area of the enterprise. Examples of business objects are Opportunity, Account, and Contact. Business objects group business components into logical units. For example, the Opportunity business object groups together the Opportunity, Contact, and Product business components, as shown in Figure 46.

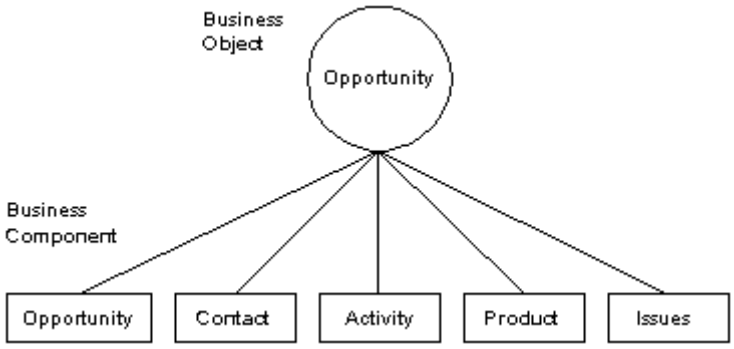


Figure 46. The Opportunity Business Object and Its Business Components

Each business object has one business component that serves as the master or driving business component. In the example shown in Figure 46 it is Opportunity. Relationships with it and other child business components (such as Contact and Product) are defined using Links, allowing the business object to display products related to an opportunity or contacts associated with the opportunity.

Business objects provide the foundation for views and screens. Typically, all the views within a screen have the same driving data for the view based upon the same business component. For example, in the Opportunity screen, the views that make up the screen include the All Opportunity List view, Opportunity Detail - Contacts view, and Opportunity Detail - Products view. The data driving these views is based on the Opportunity business component. Therefore, all the views with Opportunity–driven data are grouped into the Opportunity screen. Because all views in a screen are usually based on the same business object, a screen is indirectly related to the business object.

Business components and links can appear in multiple business objects. For example, the same two business components may have a one-to-many relationship in one business object, and the opposite one-to-many relationship (or no relationship) in another business object. However, within the context of one business object, there is an unambiguous set of relationships between the business components in the grouping.

Every view has a business object assigned to it. A master-detail view can implement only a one-to-many relationship supported by its underlying business object. For example, the view in Figure 47 can display a one contact to many opportunities relationship because Contact and Opportunity have this kind of relationship in the Contact business object, and the view (Contact Detail - Opportunities View) uses the Contact business object.

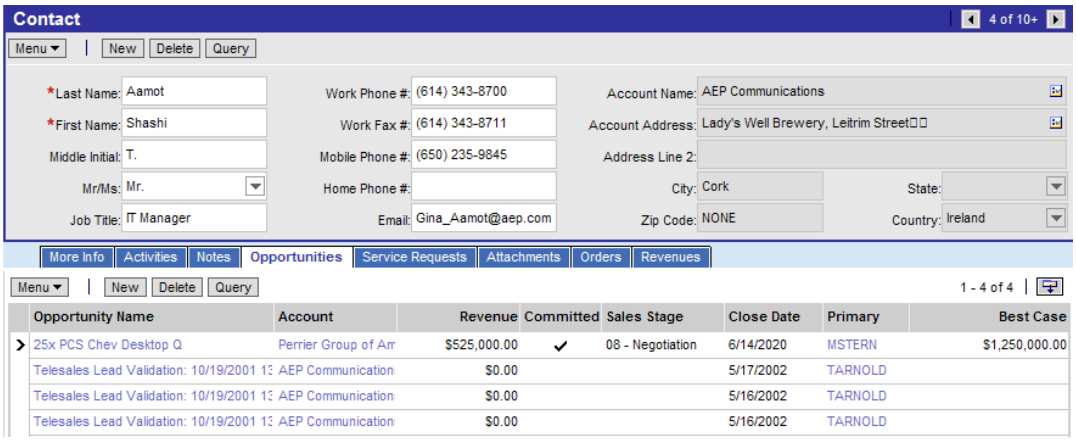


Figure 47. Master-Detail View

To implement a view displaying the reverse relationship (one Opportunity master record to many Contact detail records), the Opportunity (rather than Contact) business object would be required as the business object of the view.

Figure 48 displays the abstract relationships between the Business Object object type and Views and Screens.

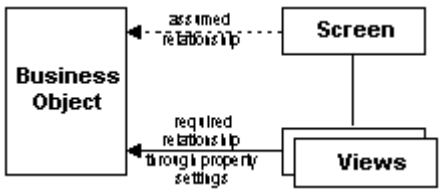


Figure 48. Relationship Between Business Object, Screen, and Views

Typically there is an 1:1 relationship between screens and business object. A business object is not assigned to a screen through a property setting the way a business object is assigned to a view. The relationship between a business object and a screen is an informal one dictated by good design practice, and it is not strictly enforced by the Siebel Tools software. In general, all of the views in a screen are associated with the same business object.

NOTE: Not all business components included in a business object participate in master-detail relationships. Business components that are not part of the business model may also be incorporated in the business object. A Business Component object makes such a business component available for use in views based on the specified business object.

How Business Objects Are Constructed

The object types illustrated in Figure 49 participate in the configuration of a business object.

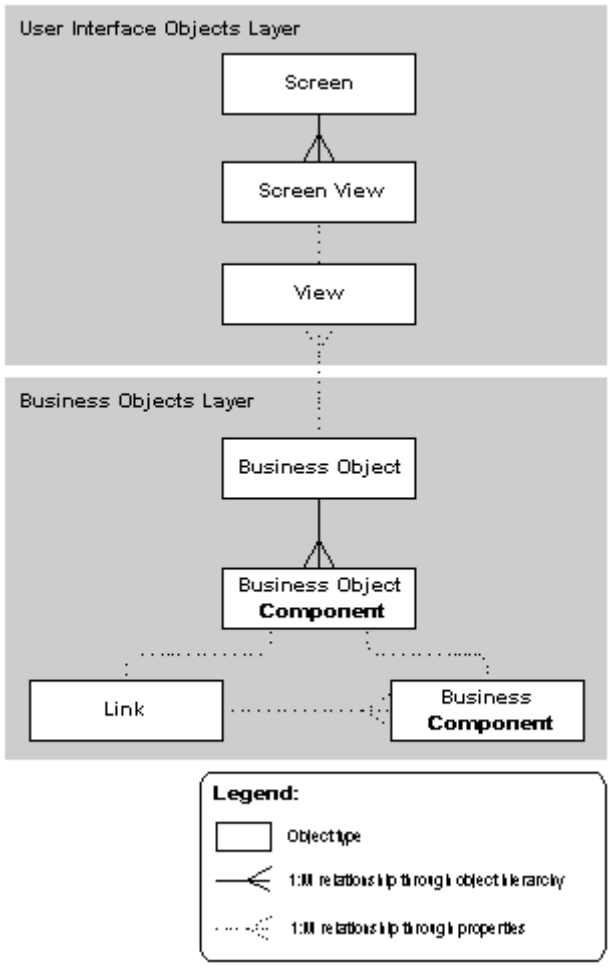


Figure 49. Business Object Architecture

The relationships between object definitions used to implement a business object appear in Figure 50.

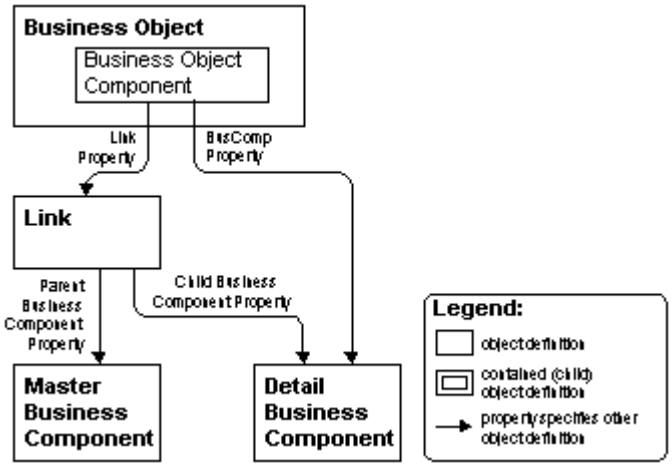


Figure 50. Business Object Details

The object definitions in Figure 50 are described as follows:

- **Business Object object type.** The business object is a parent for multiple business object component child object definitions. Each business object component specifies a master-detail relationship. View object definitions reference the business object in their Business Object property.
- **Business Object Component object type.** Business Object Component is a child object type of Business Object. Typically, each business object component defines one master-detail relationship within the parent business object. Two properties within the business object component specify this relationship:
  - **Link.** Identifies the link object definition.
  - **BusComp.** Identifies the detail business component object definition.

  A business object component can be used to include a business component in the business object without a link. To accomplish this, you enter a blank value in the Link property of the business object component. A link-free business object component allows you to incorporate a business component in the business object for use in views based on the business object, even though the business component does not have one-to-many relationships with other business components in the context of that business object.
- **Link object type.** One link is referenced by each business object component. The Link object definition specifies the master-detail relationship that is being included in the business object by way of the business object component. Links are described in "Configuring Links" on page 205.
- **Master business component.** The master business component is the "one" in the one-to-many relationship specified in the link. The Parent Business Component property in the Link object definition specifies the master business component.

- **Detail business component.** The detail business component is the "many" in the one-to-many relationship. The detail business component is specified both in the Child Business Component property of the Link object type and in the BusComp property of the Business Object Component object type.

Guidelines for Configuring Business Objects

You should rarely need to create a new business object. The only times to do so are when your design requires a new screen that groups several new business components together or groups existing business components in a way that is not supported by an existing business object. The business components that must be included in each business object are:

- Any business component whose data is displayed in an applet, on a view based on the business object.
- Any business component whose data is exported in a report, from a view based on the business object.

A business component can be included only once in each business object, and can be linked to only one other business component in the business object. In terms of the user interface, this means that applets can be linked to only one other applet in a view. Except for the Home Page view, each view has a driving applet based on the driving business component in the business object. This driving applet can have related applets based on other business components; however, these applets are always child applets of the driving applet. Therefore, all business components within the business object are either the driving business component for the business object or include data related to the driving business component. For example, to show the Contacts related to the Opportunity, the Contact business component should be part of the Opportunity business object. To show the Contacts related to an Account, the Contact business component should be part of the Account business object.

When you create a new business component to support administration or system activities, you do not need to create a new business object; make sure the new business component is part of the existing business object used to support administration views, and then assign the view to the Marketing Administration or System Administration screen.

Creating Business Objects and Business Object Components

To create a business object and business object components

1 Navigate to the Business Object object type.

2 In the Object List Editor, create a new record and then use the following table to complete the fields:

Property	Description
Name	The name of the business object must be unique among business objects in the repository. All references to the business object are done through its name.
Query List Business Component	The default value is Query List. It identifies the business component used to store predefined queries for the business object.
Primary Business Component	Defines the driving business component for the business object. You cannot define this value until after you have defined business object components.

3 In the Object Explorer, expand Business Object and select the child object type, Business Object Component.

4 In the Object List Editor, use the information in the following table to create new records for each business component you want to be represented in the business object.

Property	Description
Bus Comp	Define the business component to be included in the business object.
Link	Defines the link relationship between two business components. Although property is not required, you should define a link: ■ When the business component can be linked to more than one business component in that business object—for example, the Action business component could be linked to the Opportunity, Account, or Contact business component in the Opportunity business object. ■ When the link between the parent and child business component is many–to–many and either component can be the parent—for example, in the Opportunity business object, there is a relationship between the Opportunity and Contact business components. Because either business component could be the parent, you must specify that the Opportunity/Contact link should be used to be sure that the Opportunity business object is the parent. If you do not specify the link, by default a link named Parent Business Component/Child Business Component is used, where the parent business component property's value equals the name of the source business object, and the child business component property's value equals the value of the destination business component property. If a suitable link cannot be found, the business component is displayed without a link to any other business component in the parent business object. In this case, all records that satisfy the business component search specification independent of the parent business component are displayed. This could create issues because users would not understand that the values in the child business component are not directly related to the parent business component, but represent all data for the child business component. Therefore, you should enter a value for all links wherever you want to show master-detail records.

5 Go back to the Business Component object definition and enter a value for the Primary Business Component property.

You cannot select values for this property until after business object components have been defined.

Managing Unused Business Components

In general, any supplied unused objects must remain intact and must not be deleted, inactivated, or renamed. For business objects, this is also true. Do not delete these definitions because other objects may reference them. Delete any custom business objects that are not being used and do not reference any other object definition, such as a view.

11 Configuring Applets

Topics in This Chapter

About Applets

Applets are user interface objects that allow users to view, enter, and modify data from a single business component. They occupy a section of a view and include data controls, such as fields, text boxes, and check boxes, as well as other types of controls, such as buttons that invoke methods and ActiveX controls. Applets can be configured to display as forms, lists of records, charts, or hierarchical trees. Applets can be configured to allow data entry for a single record, to provide a scrolling table displaying multiple records, or to display business graphics or a navigation tree. Applets allow users access to the data of a single business component.

There are many styles of applets, including:

- **Form Applets.** See "About Form Applets" on page 234.
- **List Applets.** See "About List Applets" on page 235.
- **Pick Applets.** See "Types of Picklists" on page 343.
- **Multi-Value Group Applets.** See "About MVG Applets" on page 401.
- **Chart Applets.** See "About Chart Applets" on page 420.

- **Association Applets.** See "About Association Applets" on page 411.
- **Shuttle Applets.** See "About Shuttle Applets" on page 417
- **Tree Applets.** See "About Tree Applets" on page 447.
- **File Attachment Applets.** See "File Attachment Applets" on page 454.

About Applet Child Objects

The Applet object type contains several key child objects.

- **Control.** Implements controls on applets, such as text boxes, check boxes, buttons, and links. In form applets Control objects are associated with fields on the applet's business component. Controls are created as part of the New Applet Wizards.

  See "About Applet Controls and List Columns" on page 238.

- **List**. Used in List applets. A list applet has one list object definition, named List. The List object definition provides property values that pertain to the entire scrolling list table, and it serves as a parent object definition for the list column object definitions. List is automatically created as part of the New List Applet Wizard.
- **List Columns.** Child objects of List. It identifies one column in the scrolling list table and corresponds to one field in the business component. List column objects are automatically created after selecting fields in the New List Applet Wizard.
- **Applet Web Templates**. Associates an applet to a Web template. Web templates determine the layout and format of the applet when it is rendered in the user interface. An applet can be displayed in five modes. An Applet Web Template is defined for each mode. Web Templates are associated with Applet modes as part of the New Applet Wizards.

  For more information on Web templates, see "About Siebel Templates" on page 483.

  For more information about applet modes, see "About the Role of Applet Modes" on page 233.

- **Applet Web Template Items**. Child objects of Applet Web Template. Applet Web Template Items maps controls and list columns to placeholder tags or locations on a Web template. They contain the name of a control or list column as well as an identifier of a template placeholder. The placeholder determines its position of the control or list column in the Web page rendered at run time. Applet Web Template Items are automatically created when you drag and drop controls on to Web templates using the Applet Layout editor or when you create an applet using a New Applet Wizard.

Other child objects include:

- **Applet Method Menu Items**. Defines an applet-specific menu. See "Creating Applet Menus" on page 339.
- **Applet User Properties**. Allows you to configure additional functionality beyond what is available as part of the applet class. For descriptions of supported user properties for a given class, see *Siebel Developer's Reference*.

- **Drilldown Objects and Dynamic Drilldown Destinations**. Defines target views that users can drilldown from a given field. Dynamic Drilldown Destinations are child objects of Drilldowns and they are used to define several potential target views and various conditions that determine which view appears after the user clicks the drilldown. See "About Drilldowns" on page 294.
- **Tree and Tree Node**. Implements a Tree applet and all the nodes that appear when the top level of the tree is expanded. See "About Tree Applets" on page 447.

For detailed information about these object types, including property descriptions, see *Object Types Reference*.

About the Role of Applet Modes

Applets modes determine how a given applet is rendered at run time. Applets can be rendered in one or more modes. Each mode is associated with a Web template. Applet modes are defined as part of the Applet Web Template object type (child of applet) and are used when editing layouts in the Applet Layout Editor.

The modes used most often are:

- **Edit.** Used in form applets. Allows users to edit records, create new records, and query.
- **Edit List.** Used for list applets. Allows users edit records directly in the list, create new records, and query.

Other modes used are:

- **Base.** Displays fields in read-only mode. Use this mode when you want the applet to be displayed in read-only mode until the user takes an action, such as clicking the Edit button.

  You can also make views read-only using the Read-Only field in the Responsibility Administration View. For more information, see *Security Guide for Siebel eBusiness Applications*.
- **New.** Used for creating a new record where the requirements for new mode are different from the edit or edit list mode.
- **Query.** Used for querying where the requirements for the query mode are different from the edit or edit list mode.

You define modes when you create applets or when you configure applet layouts. The default mode is defined as a property on the View Web Template Item object type.

Related topics

"Creating List Applets"

"Creating Form Applets"

"Working with the Applet Layout Editor"

About Form Applets

A form applet presents business component information as a data-entry form. It allows you to display many fields for a single record. Form applets can provide a complete view of a record and are useful for data entry because users are able to access all the necessary fields at once. An example form applet is shown in Figure 51.

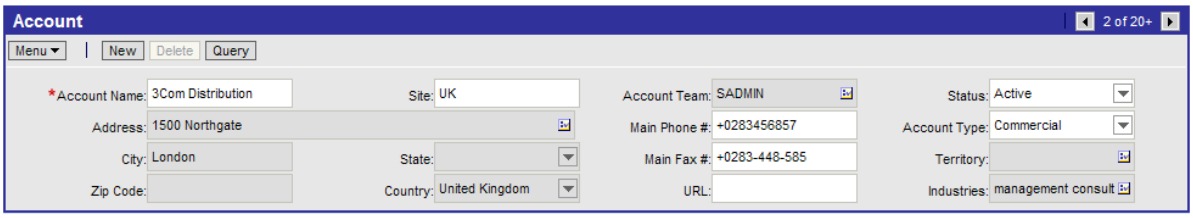


Figure 51. Form Applet

Form applets are associated to a single business component. Control objects (child of applet) associate fields on the business component to applet web template items, which are mapped to placeholders in the applet web templates associated to the applet. The relationships between object types used to implement a form applet appear in Figure 54.

NOTE: Form applets use the Control object type to expose fields in the applet. This is different from List applets, which use the List Column object type to expose fields.

The relationships between objects used to implement a form applet appear in Figure 52.

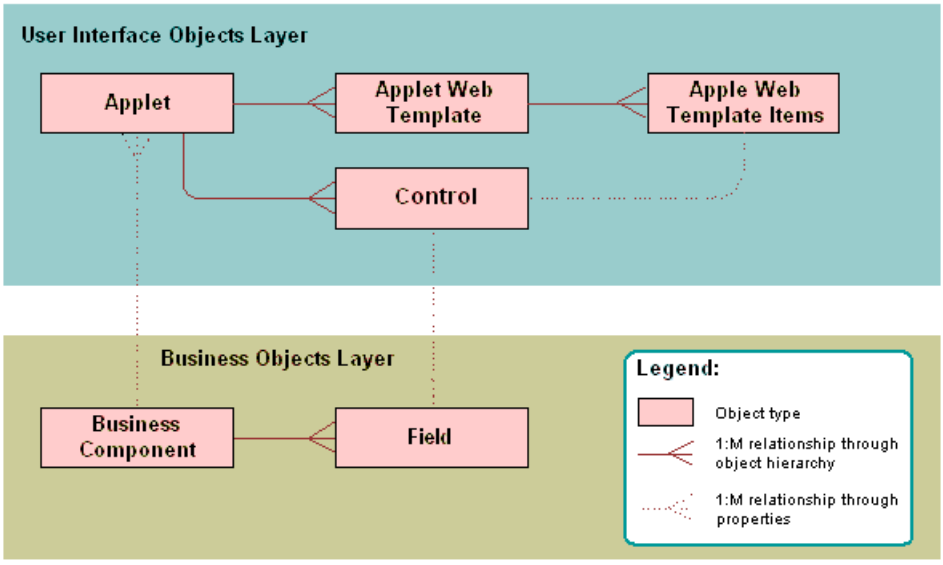


Figure 52. Form Applet Architecture

About List Applets

A *list applet* allows you to display multiple records at one time. List applets present data in table format with multiple columns. Each row of the table represents a record from the applet's business component. List applets allow users to scroll through multiple records of data and see several fields for each record. An example list applet is shown in Figure 53.

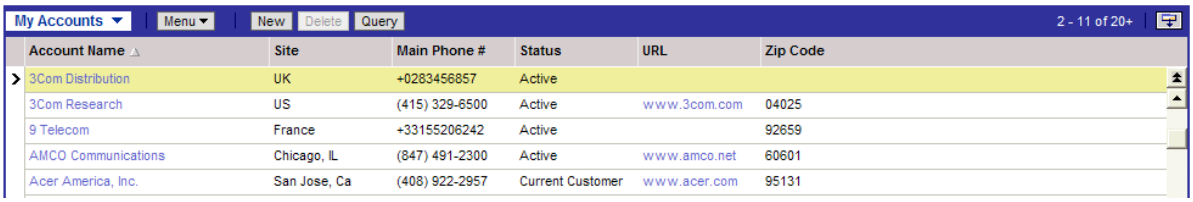


Figure 53. List Applet

List applets are associated to a single business component. The List Column child object associates fields on the business component to applet Web template items, which are mapped to placeholders in the applet Web templates associated to the applet. The relationships between object types used to implement a list applet appear in Figure 54.

NOTE: List applets use the List Column object type to expose fields in the applet. This is different from Form applets, which use the Control object type to expose fields.

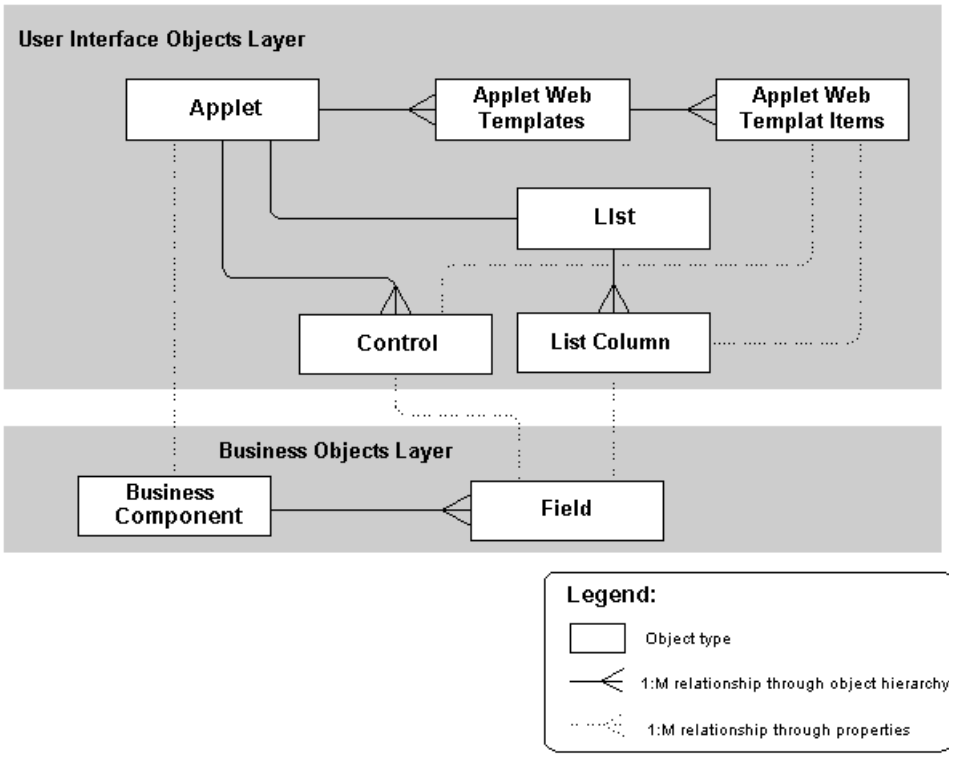


Figure 54. List Applet Architecture

Guidelines for Configuring Applets

When configuring applets consider the following guidelines:

- Keep the user interface consistent and intuitive to improve usability.
- When possible, modify existing applets instead of creating new ones. This often requires less work and helps keep the number of objects you need to maintain in the repository to a minimum.
- If you do require new applets, set the Upgrade Ancestor property on all custom applets that are cloned from other applets. For more information, see "About Upgrade Inheritance" on page 50.
- Avoid unnecessary duplication by reusing applet definitions in multiple views and screens whenever possible.
- Reduce complexity by keeping applet design simple and avoid inactive objects.

- Give duplicate applets without drilldowns the same name as the original applet but with the words *Without Navigation* immediately preceding the word Applet (or Read-Only Applet)—for example, ABC Selective Account List Without Navigation Applet.
- Applets that are used specifically for Administration purposes (which are almost always list applets) should be named <entity> Administration Applet—for example, Master Forecast Administration Applet.
- Define the required applet modes. For example, will the applet be read-only or editable? Will it have New/Query options? If the applet will be read-only mode you will only need to define it in Base mode. If it will be editable then you will have to define it in Edit and Edit List modes. If the applet will have a New/Query option and if the applet should have a different layout for the New/Query modes (compared to Edit mode) then consider creating New/Query applet Web templates.
- Do not expose unnecessary fields in list applets.

Naming Convention for Applets

The following are general naming recommendations for applets:

- Name all new applets with a prefix that identifies your company. For example, ABC Incorporated could name a new applet ABC Opportunity List Applet.
- Avoid using special characters in applet names. Use only alphanumeric characters.
- Applet names should be meaningful. Avoid adding a number suffix (for example, ABC Opportunity List Applet 2) to an applet name. For example, if the applet differs because it does not allow drill down, then indicate this in your applet name (ABC Opportunity List Applet - Without Drill Down).
- Initial-capitalize applet names, for example, Account List Applet rather than account list applet.

The type of applet should be included in the name just before the word *applet*, as shown in Table 38.

Table 38. Naming Conventions for Applets

Type of Applet	Name Format	Example
Association applets	*Xxx* Assoc Applet	Opportunity Assoc Applet
Multi-value group applets	*Xxx* Mvg Applet	Fulfillment Position Mvg Applet
Pick applets	*Xxx* Pick Applet	Order Status Pick Applet
List applets	*Xxx* List Applet	Account List Applet
Form applets	*Xxx* Form Applet (if the applet does not contain buttons) *xxx* Entry Applet (if the applet contains buttons)	Account Form Applet Account Entry Applet
Chart applets	*Xxx* Chart Applet - *yyy* Analysis [By *zzz*]	Bug Chart Applet - Severity Analysis
Tree applets	*Xxx* Tree Applet	List of Values Tree Applet

Naming Conventions for Applet Titles

Follow these general guidelines when creating applet titles:

- Always specify an applet title. Do not leave this property blank.
- No two applets in the same view should have the same title. If a view contains multiple applets displaying data from the same business component, distinguish the titles by type. For example, in a list-form view displaying accounts, use distinct titles such as *Account List* and *Account Form*.

Table 39 offers standard conventions for the titles of certain applet types.

Table 39. Title Conventions for Applets

Type of Applet	Title Format	Example
Association applets	Add *buscomp_name*(s)	Add Opportunities
Multi-value group applets	*buscomp_name*(s)	Contacts
Pick applets	Pick *buscomp_name*(s)	Pick Product
List applets	*buscomp_name* List	Account List
Form applets	*buscomp_name* Form *buscomp_name* Entry	Account Form Account Entry
Chart applets	*Xxx* Analysis or *Xxx* by *Yyy*	Open Defect Analysis Lead Quality By Campaign
Tree applets	*buscomp_name*(s)	Opportunities

About Applet Controls and List Columns

Controls and list columns are child objects of the applet object type. Controls implement user interface elements, such as text boxes, check boxes, and buttons. In form applets, controls also implement fields. In list applets, list columns implement fields.

You define controls and list columns for new applets using the New Applet Wizard. For existing applets, you add, remove, and modify controls and list columns using the Applet Layout Editor. For more information, see "Adding Existing Controls and List Columns to Applet Layouts" on page 261 and "Adding New Controls and List Columns to Applet Layouts" on page 261.

List columns, and have a corresponding object type (List Column). Data entry in a list applet is performed in the cells that are at the intersections of rows and list columns. Cells in different list columns can function in different ways, depending on the properties of their list columns. Some examples of cell behavior based on list column properties are:

- Cells in some list columns function like text controls in a form applet. This kind of cell is used for the display and editing of a text, numeric, date, or currency value. If the list column is not read-only, you can click the cell to activate an editing cursor, and edit the text.
- Cells in some list columns function like check box controls in a form applet. A check mark in the box is a TRUE value; an empty box has a FALSE value.
- When TRUE, a check box in a list column holds a check mark symbol, whereas a check box in a control in a form applet holds an X symbol.
- Cells containing underlined, colored text are *drilldown fields.* Drilldown fields let the user navigate from the cell to another view that presents detailed information about the selected row.

There are many types of controls, as defined by the HTML Type property of the control or list column object, including Text Area, CheckBox, MiniButtons, ActiveX Controls, and Links. The following topics in this section describe the various types of controls.

For a complete list of Control and List Column properties, see *Object Types Reference*.

ActiveXControls

Allows the placement of an ActiveX control in the applet.

Button Controls

Button controls are 3-d buttons that initiate an action when clicked. These type of buttons are rarely used in the application. The more common type of button is minibutton. See "MiniButtons" on page 243.

When rendered in high interactivity mode, a button can invoke a built-in method (supplied with Siebel applications), or a custom method programmed in Siebel VB, Siebel eScript, or Browser Script.

The Method Invoked property is the name of the method invoked when the button control is clicked.

There are instances when you might want to put your own custom methods in the Method Invoked property. For example, this is the only way to invoke Siebel VB, Siebel eScript, or Browser Script on a button-click event.

NOTE: The Runtime property must be set to TRUE for button controls. Otherwise the method associated with it will not execute.

To enable a button the WebApplet_PreCanInvokeMethod event must be scripted to set its CanInvoke parameter to TRUE.

eScript example:

```
function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke)

{

if( MethodName == "Map" ) {
```

```
CanInvoke = "TRUE";
return( CancelOperation );
}else {
return (ContinueOperation);
}
}
```

Check Box Controls

A check box is implemented as a control with a HTML Type property setting of CheckBox. A check box is used to represent a TRUE/FALSE field with a data type of DTYPE_BOOL. When you click an empty box, a check mark or an X appears in the box. If you click a box that is checked, the check mark or X disappears. An example of a check box appears in Figure 55.

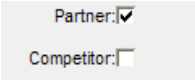


Figure 55. Check Box Control

Combo Box Controls

A *combo box* is implemented as a control with a Type property setting of ComboBox. It consists of a field with a drop-down button attached at the right edge. The user clicks the drop-down button, which activates a selection list, and then clicks a selection in the list. The selected value replaces the previous value in the box.

Combo box controls implement special-purpose picklists in chart, calendar, and pick applets. In chart applets they implement the Show and By combo boxes. In calendar applets they implement the user name combo box. In pick applets they implement the Find combo box. Combo box controls appear and behave almost identically to static picklists, but they are implemented through a different control type (ComboBox rather than Text).

A combo box control creates a UI element that allows the selection of a value from a set of values. This type can be used only if the control has a picklist defined that provides the list of values. An example of a combo box is shown in Figure 56.

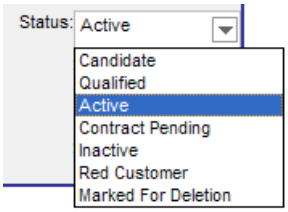


Figure 56. Combo Box

DrillDownTitle

Custom control, a title of an applet that can bring the user to the appropriate view. This is used frequently by applets on the home page. Figure 57 shows a DrillDownTitle control that provides a link to My Service Requests.

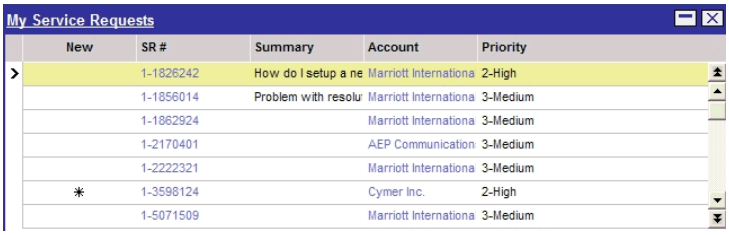


Figure 57. Drilldown Title

Field

Custom control type that has a native HTML type of Text.

FieldLabel

Custom control, a field label for a list applet.

File

Creates a user interface element that can be used to attach a file.

Hidden

Hidden controls are not visible in the Web page but can be accessed through scripting.

FormSection

A label (defined as a custom control) that helps to group related fields in an applet. The FormSection label expands to fit the region where you place it. To set it apart, the label appears against the FormSection color defined in the cascading style sheet.

NOTE: In non-grid layout form applets, the control may not expand to fit within the layout editor, but it does render correctly at run time.

Figure 58 shows an example FormSection control rendered in the user interface.

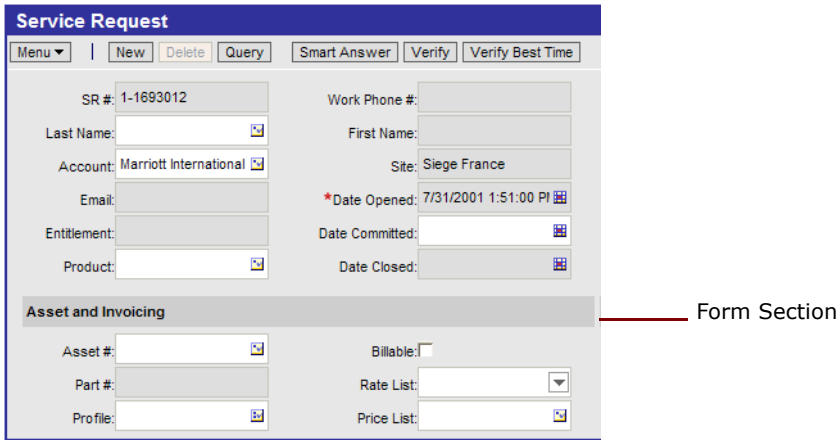


Figure 58. Form Section

ImageButton

Custom control, an image-based minibutton. See "MiniButtons" on page 243.

JavaApplet

Used for specialized applet classes and is not supported for custom configurations.

Label Controls

A *label control,* is a visual aid only. It is not tied to a business component field and has no data display or entry capabilities. Use a label control when you need to place wording somewhere inside the form applet.

NOTE: If a caption has any HTML reserved characters, such as &, <, >, ., then it should be HTML encoded as &, <, >, ", respectively.

Link

Used with controls that have an "InvokeMethod" specified (this could be a built-in method that is supplied with Siebel applications). Creates an HTML hyperlink that will invoke the method when activated. Figure 59 shows a link control “My Activities” that invokes GoToView method.

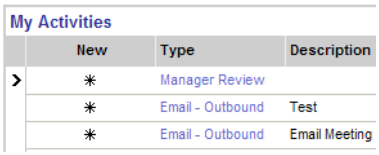


Figure 59. Link

Mailto

Used with controls that contain an email address. The control value will be displayed as a link, which when activated will open the user’s default email program with the address filled in with the control value.

MiniButtons

Custom controls that are rendered as buttons in the user interface. MiniButtons are commonly used in Siebel applications. The appearance and client-side functionality of minibuttons are defined in the CCHtmlType.swf file, located in the WEBTMPL folder of your Siebel installation folder. There are several types of minibuttons, including MiniButton, MiniButtonEdit, MiniButtonEditNew, MiniButtonEditQuery

For more information about .swf file, see “Creating Custom HTML Control Types” on page 541.

Minibuttons can be used with controls that have a Method Invoked property defined. This can be a built-in method supplied with Siebel applications or a custom method programmed in Siebel VB or Siebel eScript. When the button is clicked, the method is invoked.

The Runtime property of a button control must be set to TRUE. Otherwise the method associated with it will not execute.

To enable a button the WebApplet_PreCanInvokeMethod event must be scripted to set its CanInvoke parameter to TRUE.

eScript example:

```
function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke)

{

if( MethodName == "Map" ) {

CanInvoke = "TRUE";

return( CancelOperation );

}else {

return (ContinueOperation);

}

}
```

Password

Creates a user interface element that can be used to input a password field. The characters entered in this control will be masked by the "*" character.

PositionOnRow

Custom control that shows the currently selected record in a list.

RTCEmbedded

Custom control, an embedded text editor.

RTCEmbeddedLinkField

Custom control that allows you to display graphics and links in the RTCEmbedded object.

RadioButton

Used for specialized applet classes and is not supported for custom configurations.

RecNavNxt

Custom control used to display the next set of records.

RecNavPrv

Custom control used to display the previous set of records.

SSNxt

Custom control used to display the next question in a SmartScript.

SSPrv

Custom control used to display the previous question in a SmartScript.

Text

A *text control* displays text inside a rectangular box. An example of a text control is Name, shown in Figure 60.

Figure 60. Sample Figure

Some characteristics of Text controls are as follows:

- A text control allows the entry and editing of text, unless the Text control is read-only (in which case it has a gray background, and displays text which cannot be altered).
- A text control displays data of a particular data type, such as alphanumeric, numeric, date, or currency and will display as drop-down list, picklist, MVG, calculator, or calendar icon depending on the underlying business component field.
- A Select icon is automatically attached to the right edge of a Text control when the MVG Applet property has a nonblank value or the Pop-up Edit property is TRUE. This allows the user to call up a multi-value group applet or a calendar or calculator widget.

  NOTE: The Runtime property must also be TRUE any time the field is supposed to pop up a Calendar or Calculator type control.

- The Select button is attached to the right edge of a text control when the Pick Applet property has a non-blank value. This allows the user to call up a picklist by clicking the icon.

  Static picklists and pick applets are discussed in "About Static Picklists" on page 344.

NOTE: Trailing spaces are truncated in data displayed through the Siebel application user interface or through Siebel Tools.

When configuring controls of type Text the consider the following:

- **Field.** The field in the business component from which the Text control displays data.
- **Display Format.** A format specification for data displayed by the Text control, used for numeric, date, currency, and similar non-text data types. Used as follows:
  - For DTYPE_NUMBER data, the property can be left blank (indicating that the appearance of numeric values should be as set in the Regional Settings section of the Windows Control Panel) or explicitly specified using 0, #, +, minus sign, comma, and period symbols.
  - For DTYPE_CURRENCY data, the property can be specified explicitly using the same symbols as for DTYPE_NUMBER, plus the dollar sign. The display of currency values can also be controlled using the Scale field in the Currencies view under the Application Administration screen.
  - For DTYPE_DATETIME data, one of the keywords Date, Date Time, Date TimeNoSec, and TimeNoSec may be specified.
  - For DTYPE_DATE data, the property can be left blank (indicating that the appearance of date values should be as set in the Windows Control Panel) or explicitly specified using combinations of M, D, Y, and / symbols.
  - For DTYPE_TIME data, the keyword TimeNoSec can be entered, the property may be left blank (indicating that the appearance of time values should be as set in the Windows Control Panel), or a format mask may be explicitly specified using combinations of H, h, m, s, and : symbols.
  - For DTYPE_PHONE data, the Display Format property is left blank, and the Windows Control Panel setting is used.

  NOTE: Postal code formatting options are not explicitly provided, and hyphens in a postal code are not supported. Generally, for postal codes you should use the DTYPE_NUMBER data type, and a format mask in the Display Format property consisting of octothorpes and blank spaces, such as ##### #### for U.S. Zip+4 postal codes.
- **Read Only.** A TRUE/FALSE value. Indicates if the user can edit the value displayed in the text box.

  NOTE: The Read Only property must be set to FALSE to use the Runtime property to access multi-value groups and pick applets.

- **Runtime.** This is a TRUE/FALSE value. When the text box control has an MVG Applet or Pick Applet property setting other than blank, a value of TRUE in the Runtime property directs the system to activate an icon or drop-down arrow to the right of the text box. A FALSE value directs the system not to provide the icon or arrow. This makes the multi-value group or pick applet inaccessible.

  NOTE: A Runtime setting of TRUE, combined with blank MVG Applet and Pick Applet property settings, directs the system to determine from the data type of the underlying field if an icon for a calculator, calendar, or currency pop-up applet should be provided.

- **MVG Applet.** Identifies the applet to use for the multi-value group dialog box (multi-value group applet). The field for the control must be a multi-value field, and the Runtime property must be set to TRUE.
- **Pick Applet.** Identifies the applet to use for the picklist dialog box (pick applet). The field for the control must have a picklist specified, and the Runtime property must be set to TRUE.

TextArea

Used to create a user interface element that can be used to enter text in multiple lines. An example TextArea control is shown in Figure 61.

Figure 61. TextArea

URL

Used with controls that contain URL values. The value will be displayed as a hyperlink, which when activated will take the user to the URL.

About Run-Time Pop-Up Controls

For controls or list columns of type Text or Field, Siebel applications enable specialized controls, such as a calendar or a calculator, to pop up at run time, depending on the field data type. Table 40 summarizes the data types and associated pop-up controls.

Table 40. Run-Time Pop-Up Controls

Field Data Type	Pop-Up Control
DTYPE_DATE	Calendar
DTYPE_TIME	Time

Table 40. Run-Time Pop-Up Controls

Field Data Type	Pop-Up Control
DTYPE_DATETIME	Combination calendar/time
DTYPE_NUMBER	Calculator
DTYPE_INTEGER	Different pop-up controls based on what is configured for standard interactivity.

To cause a run time pop-up control to appear, the Runtime property of the list column or control must be set to TRUE.

If there is a picklist defined for a field that has one of the types in Table 40, then a picklist pops up at run time, instead of a calculator or calendar.

NOTE: The Read Only property of the list column or control must be set to FALSE to use the Runtime property to access run-time pop-up controls.

Guidelines for Configuring Controls and List Columns

When configuring controls or list columns consider the following guidelines:

- Use the appropriate pop-up device wherever possible to ease data entry. For instance, you should associate a calendar control with a date field, and a multi-line edit box with a multi-line text field.
- In forms and lists, fields should be left aligned, except for fields displaying numeric values such as currency. These fields should be right aligned. In forms, labels should be right-aligned.
- When creating check boxes:
  - Consider whether a check box is the appropriate way to display the data. In situations where the data does not map well to a yes or no response, or where the meaning of the unchecked value is not obvious, it is better to use a list of values. For example, instead of a check box labeled Standard, use a combo box labeled Shipping Method, with a list of values containing Standard and Next Day.
  - Avoid negatives. For example, instead of *Not Required,* use *Optional*.
  - In form applets, position labels to the left of the check boxes and right-align.
- For Text controls and list columns:
  - If a calendar or calculator needs to appear for a control, set the Runtime property of the control to TRUE.
  - If a pop-up editor needs to appear, set the Popup Edit property of the control to TRUE.
  - If you want to make a list column unavailable in the user interface, set the Available property to FALSE.

- If you want a list column to be displayed in the list by default, set the Show in List property to TRUE. However, if you want the list column to be hidden by default, but you want end users to be able to select a list column to be displayed using the Columns Displayed dialog box, set the Show in List property to FALSE.

- Display Names and Captions may need to include extra spaces to accommodate multiple translations of a given string. For more information about strings, see *Using Siebel Tools*.
- Controls for form applets using the "Applet Form 4 Column (Edit/New)" Web template can be associated to either a "2-Column Wide field" or a "1-Column Wide field". To associate a control to a "2-Column Wide field", you must set the HTML Width property to 412. If you do not specify an HTML Width property, the control will appear as a "1-Column Wide field" even when it is associated to a "2-Column Wide field" on a form applet.

Creating List Applets

You create list applets using the List Applet Wizard. The List Applet Wizard helps you identify all the correct properties and automatically creates child objects, such as Web Template Items, based on the information you enter. You can also create applets manually by defining all the necessary properties and child objects.

The List Applet Wizard does the following:

- Creates the list applet
- Creates the applet Web template
- Creates the list, list columns, and controls
- Creates applet Web template items

NOTE: The List Applet Wizard can not be used when the ClientConfigurationMode CFG parameter is set to All.

To create a list applet using the List Applet Wizard

1. From the application-level menu, choose File > New Object.

   The New Object Wizard dialog box appears.

2. Click the Applets tab, and then double-click the List Applet icon.

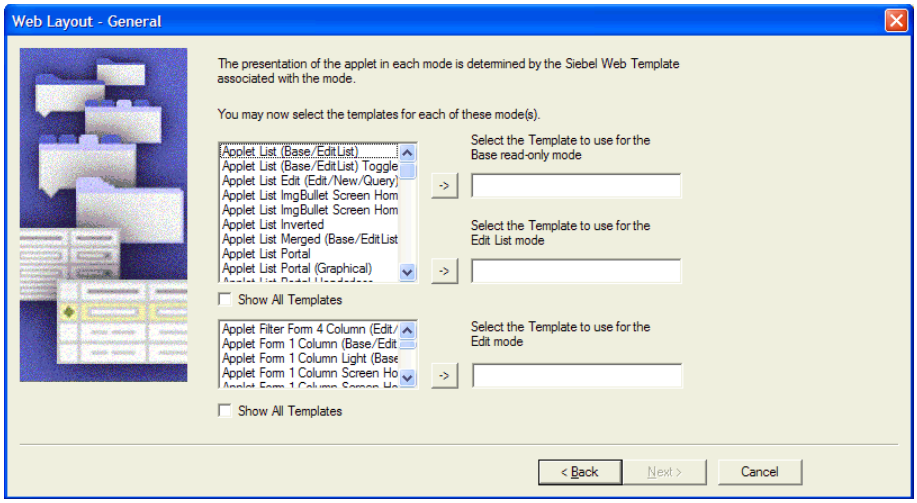
   The General page of the List Applet Wizard appears.

3 In the General Page, enter the following information for the applet, and then click Next.

Field	Example Value	Comment
Project	Account	Only locked projects appear in the picklist.
Business Component	Account	The business component that the applet is based on.
Applet Name	New Account List Applet	A unique name for the Applet.
Display Title	Accounts	The name to appear in the user interface.

The wizard will use information to create an applet object and define the required applet properties.

4 In the Web Layout-General page, enter the Web templates to use for each mode of the applet, and then click Next.



The Web templates initially available in the wizard are filtered based on Web Template Type. To display all templates, select the Show All Templates check box.

A thumbnail image for most templates appears when you select the template name. For a complete description of all templates, see *Siebel Developer's Reference*.

5 In the Web Layout - Fields page, select the fields that you want to appear on the applet, and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in Step 3 on page 250.

6 In the second Web Layout-Fields page, choose the controls in the Available Controls box that you want to appear on the applet, and then click Next.

   All the entries in the Selected Controls box are added by default. The available controls come from the *Model HTML Controls* applet. This applet specifies the available controls and to which template each control is mapped. You can modify the model applet if necessary by adding or removing controls from the applet.

7 Review the information displayed in the Finish page, and then click Finish.

   The List Applet Wizard creates the applet and supporting object definitions based on the selections you made.

   The Applet Layout Editor opens and displays the layout of the new list applet ready for you to edit.

   See "Working with the Applet Layout Editor" on page 259.

   NOTE: You can return to previous pages by clicking the Back button.

Creating Form Applets

You create form applets using the Form Applet Wizard. The Form Applet Wizard helps you define the correct properties and automatically creates child objects, such as Web Template Items, based on the information you enter.

The Form Applet Wizard does the following:

- Creates the form applet
- Maps the applet to an applet Web template
- Creates the controls
- Maps controls to Web templates by creating Applet Web Template Items

NOTE: The Form Applet Wizard can not be used when the ClientConfigurationMode CFG parameter is set to All.

To create a Form Applet using the Form Applet Wizard

1 Choose File > New Object from the Siebel Tools main menu.

   The New Object Wizard dialog box appears.

2 Click the Applets tab, and then double-click the Form Applet icon.

   The General page of the Form Applet Wizard appears.

3 In the General Page, enter the following information for the applet.

Field/Check Box	Example Value	Comment
Project	Account	Only locked projects appear in the picklist.
Applet Name	New Account Form Applet	A unique name for the Applet.
Business Component	Account	The business component that the applet is based on.
Display Title	Accounts	The name to appear in the user interface.
Use Grid Layout	Check mark	Selected by default. Siebel Systems recommends that you configure form applets using grid-based templates. Grid-based templates allow you to control the layout of the form applet using the Applet Layout Editor. For more information, see "About Grid Layout" on page 266.

The wizard will use this information to create an applet object and define the required applet properties.

4 Do one of the following:

- If you selected Use Grid Layout on the previous dialog, choose whether or not you want the applet to display in Base mode. The appropriate Web template is automatically selected for Edit Mode.
- If you did not select the Use Grid Layout option on the previous dialog, select the Web templates you want to use for each mode.

NOTE: In most cases, using Edit mode only is sufficient. However, you may want to use base mode too. For example, this is useful when you want the applet to be read-only until the user takes an action, such as clicking the Edit button. Or you may want to use Base mode only. Only one mode is required.

5 In the Web Layout - Fields page, select the fields that you want to appear on the applet and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in Step 3 on page 250.

6 In the second Web Layout-Fields page, choose the controls that you want to appear on the applet and then click Next.

All the entries in the Selected Controls box are added by default. The available controls come from the *Model HTML Controls* applet. This applet specifies the available controls and to which template each control is mapped. Users can modify this applet if necessary by adding or removing controls from the applet.

7 Review the information displayed in the Finish page, and then click Finish.

The Form Applet Wizard creates the applet and supporting object definitions based on the selections you made.

The Applet Layout Editor opens and displays the layout of the new list applet ready for you to edit.

NOTE: You can return to previous pages by clicking the Back button.

Adding Web Templates to Applets

You add Web templates to applets using the Object List Editor. You need to add templates to applet when you want to define additional modes for applets.

To add a Web template to an applet

1 Navigate to the applet you want to modify.

2 In the Object Explorer, select the Applet Web Template object type (child of Applet).

3 In the Applet Web Templates list, right-click and choose new record.

4 Use the information in the following table to complete the record:

Property	Description
Name	By convention this field is used to describe the mode of the Applet Web template.
Type	Applet Web template mode.
Web Template	Select Web template to associate to the applet.

Related Topic

"About the Role of Applet Modes" on page 233

Setting Applet Search Specifications

If the value in the Search Specification property in an Applet object definition is nonblank, the set of records provided to an applet is restricted. The search specification contains the names of one or more fields in the business component and various operators, combined to create a conditional expression. Records in which the value of the conditional expression evaluates to TRUE are provided to the applet for display; those records in which the expression evaluates to FALSE are excluded.

Search specifications on child applets are not executed when the child applet is based on the same business component as the parent applet. When that is not the case, search specifications on child applets are executed. They are amended with a WHERE clause that keeps the search specification in context with the parent applet.

Some sample search specification expressions appear below:

```
[Type]= "COST LIST"

[Revenue] > 5000

[Competitor] IS NOT NULL and [Competitor] <>  "N"

[Type] = LookupValue ("TODO_TYPE", "In Store Visit")
```

Search specification expressions are built according to the following syntax rules:

- Standard comparison operators are used to compare a field to a constant, or one field to another field. These include =, <>, >, <, >=, and <=.

  Example: `[Revenue] > 5000`

- String constants are enclosed in double quotation marks. String values are case sensitive, so the use of uppercase and lowercase letters in the search specification should exactly match that of the records you want returned.

  Example: `[Type] <> “COST LIST”`

- The logical operators AND, OR, and NOT are used to negate or combine expressions. Case is ignored in these operators; for example, “and” is the same as “AND”).

  Example: `[Competitor] IS NOT NULL and [Competitor] <>  “N”`

- A field name in a search specification must be enclosed in square brackets.

  Example: `[Conflict Id] = 0`

- The LIKE operator can be used to create text string comparison expressions in which a field is compared to a constant, or a field to another field, and a match on only the first several characters is required. The wildcard characters “*” and “?” are used to indicate any number of characters, and a single character, respectively.

  Example: `[Last Name] LIKE “Sm*”`

  In this example, the Last Name values of Smith, Smythe, Smallman, and so on would cause the expression to evaluate to TRUE.

- The search specification expression must be 255 characters or less.

An applet search specification cannot be used to override the search specification of the underlying business component, if the business component has one. Rather than overriding the business component’s search specification, the applet’s search specification is appended to that of the business component. Search specifications should appear in the business component or the applets that use it, but not both.

The search specification on an applet is converted to a WHERE clause by the data manager at run time. When two applets based on the same business component appear in the same view, one query is generated against the database to populate both applets. Because a database select statement only supports one WHERE clause, only one of the applets should have a search specification—or if both do, they should have the same specification.

For example, the Account List Applet and the Account Entry Applet both appear in the Account List View. The record that is selected in the Account List Applet also appears in the Account Entry Applet. When you select a different row in the list or scroll through the list, the Account Entry Applet is updated to show the same record that is selected in the Account List Applet. This is made possible by the fact that both applets are populated from the same query and therefore show the same record set.

To prevent the two applets from being synchronized, they would have to be on separate business components, for example by copying the business component on which the first applet is based.

For more information on the usage of the Search Specification property of applets, see *Object Types Reference*.

When the Applet Visibility Type property of the View Web Template Item object is set to a non-null value, it might cause search specifications on the applets in that view to be ignored. This property is recommended for use mainly where the applets in a view are based on different business components. If you use this property, test it thoroughly for functionality.

Search specifications can affect performance negatively, particularly when you include fields based on joins in the search specification. Search specifications with NOT or OR can also adversely affect performance by forcing the database to execute a full table scan.

For more information about performance, see *Performance Tuning Guide*.

Exposing System Fields

To expose system fields in the user interface, you create either a list column (within a list applet) or a control (within an entry applet).

NOTE: You do not need to define system fields as child Field objects of the underlying business component.

To define a list column object

1. Create a new list column object.
2. Enter a list column field property of *xxxx* where *xxxx* represents the system field that maps to the system columns. Table 41 lists the system fields.

Table 41. System Fields

Field	Description
Updated	System date and time the record was last updated.
Updated by	Login ID of the person who last updated the record.
Created	System date and time the record was initially created.

Table 41. System Fields

Field	Description
Created by	Login ID of the person who initially created the record.
Id	Row ID of the record.

3 Enter a display name property for the list column (for example, Last Updated).

Setting Default Method for an Applet

The default method is the one that is invoked when the user presses Ctrl+Enter. For applets in query mode, this is ExecuteQuery (pressing Alt+Enter will also execute the query). For other modes, the DefaultMethod applet user property can be set.

NOTE: This must be a valid applet InvokeMethod, such as NewRecord or GotoNextSet.

To set the default method for an applet

1 In Siebel Tools, select the Applet object, and then select the desired applet.

2 Expand the Applet object, and then select the Applet User Prop object.

3 Add a new record with the name DefaultMethod and the value of the method you want to be invoked.

Changing Styles of Label Text

You can change the style of text strings that are stored in the Caption property of a control and the Display Name property of list columns by embedding HTML tags in the property values.

For example, a Caption property with the value of `<font color="red" size=+2><b>Account Name</b></font>` would be rendered at run time with changes that reflect the HTML tags.

You cannot use HTML tags in string-based properties, such as List Columns, that are interpreted as literal values by SWE when rendered in high interactivity mode.

There are restrictions to what HTML tags you can use.

- HTML tags that control text style, such as size, color, italics, and bold, are supported.
- Other HTML tags, such as those that control alignment or position, are not supported.

NOTE: To modify text strings by adding HTML tags, you must either create a new symbolic string that includes the HTML tags or enter the values in a string-override field. To be able create symbolic strings or string override fields, you must have the EnableToolsConstrain parameter of the `tools.cfg` file set to FALSE. For more information about working with symbolic strings, see *Using Siebel Tools*.

12 Editing Applet Layout

Topics in This Chapter

Working with the Applet Layout Editor

The Applet Layout Editor is a visual editing tool that allows you to modify applet layouts, including adding and removing controls and list columns. It provides a layout canvas to work in and a preview mode that allows you to see how the applet will be rendered at run time.

Before working in the Applet Layout Editor, be aware of the following settings:

- **Language mode.** The current language of Siebel Tools (displayed in the lower right corner of the Siebel Tools window). It allows you to work with locale-specific data, such as translatable text strings, in languages other than English. The language mode also determines the locale-specific records that are transferred during check in and check out and compiled to the .srf. You can change the language mode by choosing View > Options, then selecting the Language Settings tab.
- **Constrain mode.** The EnableToolsConstrain parameter in the tools.cfg file determines the constrain mode. When it is set to TRUE, you must choose from a list of string references to enter values for translatable text strings, such as an Applet Title, and you cannot create new symbolic strings. When set to FALSE, you can override the string reference using the string override property. You can also create new symbolic strings. The Constrain Mode is defined by the `EnableToolsConstrain` parameter in the tools.cfg file.

For more information about the language mode or the constrain mode, see *Using Siebel Tools*.

To edit applet layouts

1. Select a Target Browser from the drop-down list in the Configuration Context toolbar.

   If you do not select a browser, an error message appears when you open the Applet Layout Editor.

2. In the Object Explorer, select the Applet object type.
3. In the Object List Editor, select an applet whose layout you want to modify.
4. Right-click, and then choose Edit Web Layout.

   The Applet Layout Editor appears.

   NOTE: If an applet Web template is not yet associated with the applet you selected, a dialog box appears that allows you to open the Applet Wizard to associate a Web template with the applet.

5. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

   Make sure that you select an active Web template. Both active and inactive Web templates are displayed. Inactive Web templates are labeled as such.

   Changes to an applet layout in one mode are not automatically propagated to the applet layout in another mode.

6. In the Application field of the Configuration Context toolbar, choose an application.

   Choose All Applications when you want configuration changes to apply to all applications. Choose a specific application when you want configuration changes to apply to the selected application only. For more information about configuring for a specific application, see "About Application-Specific Mappings" on page 277.

7. Edit the applet layout as necessary.

   If you are adding new controls or list columns to the applet layout, you can define object properties, such as Field and Name, using the Properties window.

8. Save your changes to the Web Layout by choosing File > Save.

Adding Existing Controls and List Columns to Applet Layouts

In the Applet Layout Editor, you can add existing controls and list columns to the applet layout. Existing controls and list columns are child objects of the applet object that are already created in the repository but have not been mapped to the applet Web template.

For more information about controls and list columns, see "About Applet Controls and List Columns" on page 238.

To add existing controls and list columns to applet layouts

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of the Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Application field of the Configuration Context toolbar, choose an application.

   Choose All Applications when you want configuration changes to apply to all applications. Choose a specific application when you want configuration changes to apply to the selected application only. For more information about configuring for a specific application, see "About Application-Specific Mappings" on page 277.
4. From the Controls/Columns window in the Applet Layout Editor, select the controls or list columns that you want to add to the layout.

   If the Controls/Columns window is not displayed, select View > Windows > Controls Window.
5. Drag the controls or list columns and drop them onto the applet layout.

   The controls and list columns appear in the layout canvas and the corresponding Web Template Items are automatically created.

   NOTE: In applets based on grid layout applet Web templates, labels and controls are treated as separate items to provide more flexibility when designing the form layout. However, this requires you to map both the control and its corresponding label onto the applet layout. Note that a label has the same name as its corresponding control, except that it is appended with the word *label*. For more information, see "About Grid Layout" on page 266.

Adding New Controls and List Columns to Applet Layouts

In the Applet Layout Editor, you can add new controls and list columns to an applet layout. For example, you can add a new control to the applet layout that exposes a custom business component field. When you add controls, the Applet Layout Editor automatically creates the corresponding child objects (Controls, List Columns, and Applet Web Template Items) for the applet.

To add new controls or list columns to applet layouts

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.

2 In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

3 In the Application field of the Configuration Context toolbar, choose an application.

   Choose All Applications when you want configuration changes to apply to all applications. Choose a specific application when you want configuration changes to apply to the selected application only. For more information about configuring for a specific application, see "About Application-Specific Mappings" on page 277.

4 On the Web Controls toolbar, click the icon representing the control you want to add, and then click the position in Applet layout where you want it to appear.

5 The Control or List Column object definition is automatically created and mapped to a Applet Web Template Item.

6 Choose File > Save.

7 Define the properties for the control using the Properties Window.

   If the Properties window is not open, select the control, right-click, and then choose View Properties Window.

NOTE: There is a placeholder in the header for list and form applets that allows you to add a button and be rendered only in SI mode. To see this placeholder (which has an item ID of 580), on the Configuration Context toolbar, choose the Interactivity drop-down list, and then select Standard.

Positioning Controls and List Columns in Non-Grid Layouts

For applets based on non-grid layout Web templates, such as list applets, the locations available for controls are determined by predefined placeholders in the Web template. For these applets, you position controls by dragging and dropping them onto empty placeholders displayed in the layout canvas.

To position controls on an applet layout that is not grid-based

1 Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.

2 In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

3 From the Controls/Columns Window, drag and drop the controls or list columns to any empty placeholders displayed in the layout canvas.

   Applet headers and footers are designed for button controls. Avoid mapping non-button controls, such as fields, to applet headers and footers.

Deleting Controls and List Columns

You can cut or delete controls from an applet layout without having to delete the corresponding Control and List Column object types.

CAUTION: When working in language override mode, do not delete controls or list columns from applet layouts. Instead, set the Visible property to FALSE. If you delete controls or list columns when working in language override mode, the corresponding items will be deleted for all languages, not just the one in which you are working.

To delete controls

- Select the controls you want to delete and do one of the following:
  - Choose View > Cut or pressing Ctrl +X.
  - Right-click, and then select Delete.

  The items are removed from the layout canvas and the corresponding applet Web template items are deleted from the repository.

You can select multiple controls by holding down the Shift key and then selecting the controls you want to delete.

Editing List Columns Display Names and Control Captions

You can edit the Display Name property of list columns and the Caption property of controls directly in the Applet Layout Editor. When you type in a string, Siebel Tools searches for a symbolic string with a matching value in the Current String Value property. If an exact and unique match is found, the symbolic string is associated with the control or list column and the current string value is entered in the Display Name or Caption field. If no record is found, or there is not a unique match to a single symbolic string, an error is displayed that prompts you to enter a new symbolic string in the Object List Editor. You can also enter a value in the string override field in the Properties window. To be able to create new symbolic strings and enter string override values, you must be working with the ConstrainedMode parameter in the tools.cfg file set to FALSE.

For more information about working with symbolic strings, string override fields, and the ConstrainedMode parameter, see *Using Siebel Tools*.

To edit list column display names and control captions

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the layout canvas, double-click a control or list column.

4 Highlight the display name or caption and then type new text.

Siebel Tools searches for a symbolic string that has a Current String Value property that matches what you typed.

After you save your work, the Display Name property for list columns and the Caption property for controls are automatically changed.

NOTE: For grid-based templates, you need to change the Caption value of a label by editing it in-line on the label itself or by selecting the field and editing the field's (and thus the control's) properties. Labels get their property values from their corresponding controls. See "Working with the Applet Layout Editor" on page 259 for more information about labels and controls in grid-based applet Web templates.

Displaying a Control When the Show More Button Is Selected

You can configure controls so they are displayed in the user interface only after the Show More Button is selected.

To configure a control for Show More button

1 Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.

2 In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

3 In the Layout Canvas, select the control, right-click, and then choose More.

The control appears in the layout canvas with an arrow icon preceding it. At run time, control will appear only after the user selects Show More button.

See "About Application-Specific Mappings" on page 277.

Previewing the Applet Layout

You can preview the applet layout to see how it will look after being rendered at run time. The Applet Layout Editor allows you to display the layout as it would look when displayed under various conditions at run time. For example, it allows you to display the layout:

- In different applet modes
- In high interactivity mode or standard interactivity mode
- For a specific application

When working with the preview mode, consider the following:

- Hide any docked windows, such as the Object Explorer or the Properties window, to see the layout rendered in full view.

- A grid is displayed in the preview mode that allows you to estimate the width of the applet layout. The default grid consists of cells that measure 100 by 100 pixels. A red bar in the preview mode indicates the right edge of the layout, beyond which users of the run time application will have to scroll horizontally. The bar is 2 grid-cells wide. For the default grid, the bar appears at 969 pixels which is optimized for a display resolution of 1024.

  You can change the preview mode background grid to optimize layouts for different monitor settings. For more information about changing the default grid, see *Using Siebel Tools*.

- When the layouts of grid-based applets are displayed in preview mode, the spaces between fields and the spaces in labels may be compressed. However, fields are never compressed.

To preview applet layouts

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. Click anywhere on the layout canvas to select it, right-click, and then choose Preview.

   The applet is displayed as it would be rendered in the user interface at run time.

Export the Preview to an HTML File

After previewing an applet layout, you can export it to an HTML file for later viewing.

To export the preview displayed in the Applet Layout Editor

1. While in Preview mode, select File > Export.
2. In the dialog box that appears, choose a file name and define *siebel_tools_installation*\tools\public\enu as the directory.

   You need to specify this directory so that image files, such as buttons, are rendered correctly in the HTML file.

Checking Mappings

Sometimes mappings between Controls or List Columns and placehoders in Web templates can become invalid. This may occur when a Control or List Column object is inactivated or is deleted from the repository, but still appears on the Web template. It may also occur when a new Web template is associated with an applet, and the existing placeholder IDs for Controls or List Columns do not exist in the new template.

To check mappings

1. In the Applet Layout Editor, click the Layout Canvas to select it.

2 Right-click, and then choose Check Mappings.

If mappings between controls or list columns and placeholders are invalid, you are prompted to delete it from the Web template.

About Grid Layout

Grid layout is a design technology used in the Applet Layout Editor and supporting applet Web templates that allow you to modify form applet layouts without having to directly modify the underlying applet Web templates. The work space is a grid-based canvas where controls snap to a grid of cells that measure 8 x 8 pixels each. You can configure the layout of form applets using a palette of layout tools, such as resizing, aligning, and centering.

To support the grid layout features in the Applet Layout Editor, applets must be based on one of the grid layout applet Web templates (AppletFormGridLayout or AppletPopupFormGridLayout). Most form applets are based on grid layout Web templates. You can determine if an applet uses a grid-based applet Web template by looking at the Web Template property of the Applet Web Template object type (child object of Applet). For more information, see "About Form Applet Templates (Grid-Based)" on page 497.

Working With Grid Layout

When working with applets based on grid-layout templates, consider the following:

- Controls snap to a grid in which each grid cell measures 8 x 8 pixels.
- Grid-based layouts only partially resize fields based on the user's monitor settings. For example, if a grid-based form designed to run on a monitor that is set to a resolution of 1024 x 768 (full width) is displayed on a monitor that is set to 800 x 600, the user would have to scroll to the right to see the right edge of the layout. If the same form is displayed on a monitor set to a resolution of 1280 x 1024, it would not occupy the entire width of the screen.

  However, the browser will eliminate as much padding as possible to eliminate the potential for horizontal scroll to occur. Field sizes do not change, but blanks at the start or end of labels will. For example, an applet designed to be 150 grid cells wide (1200 pixels), could actually render within the width of the screen at 1028 resolution. The Tools Preview mode will reflect this. You can use the preview mode to approximate how wide the form will be when it is rendered in the browser by tracking how many 100pixel-wide cells it crosses.
- In the Applet Layout Editor, labels appear as separate items from their corresponding controls. This gives you the ability to independently configure the layout of labels. Note that labels are used as constructs in the Applet Layout Editor only. Labels do not exist as separate controls in the Siebel repository, yet they are defined as applet Web template items. They use the Caption and Text Alignment- Label property of their corresponding control, but other properties do not apply.
- When you save an applet layout, the Applet Layout Editor checks for overlapping controls. If overlapping controls exist, an error message appears and you cannot save the layout.

- When using the alignment buttons on the Format toolbar, the last item selected on the canvas will be the one to which all other items will be aligned, centered, spaced, and so on.

Positioning Controls in a Grid Layout

Grid layout allows you to position controls anywhere in the applet layout.

NOTE: Do not resize or position controls by modifying object properties in the Object List Editor. Instead, always use the layout editor.

To position controls

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Layout Canvas, select one or more controls.
4. Do one of the following:
   - Drag and drop the control or controls to the desired position.
   - Use the arrow keys to move the control or controls to the desired position.
5. Choose File > Save.

Aligning Controls in a Grid Layout

Grid layout allows you to align controls relative to each other.

To align controls

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Layout Canvas, hold down the Shift key, and then click the controls you want to align.

   The last item selected is the one to which all other items will be aligned.
4. Perform the following tasks as necessary:

Task	Button
Align the left side of controls.	
Align the center of controls vertically.	

Task	Button
Align the right side of controls.	
Align the top edge of controls.	
Align the middle of controls horizontally	
Align the bottom edge of controls	

5 Choose File > Save.

Making Controls the Same Size in a Grid Layout

Grid layout allows you to make Controls or List Columns the same size as other Controls and List Columns.

To resize controls

1 Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.

2 In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

3 In the Layout Canvas, hold down the Shift key, and then click the controls you want to resize.

The last item selected is the one to which all other items will be resized.

Perform the following tasks as necessary:

Task	Button
Make selected controls the same width.	
Make selected controls the same height.	
Make selected controls the same size.	

4 Choose File > Save.

Spacing Controls in a Grid Layout

Grid layout allows you to control the horizontal and vertical spacing of controls relative to each other.

To change the space between controls

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Layout Canvas, hold down the Shift key, and then click the controls whose spacing you want to modify.

   The last item selected is the one that is used as the basis for the spacing.
4. Perform the following tasks as necessary:

Task	Button
Make horizontal spacing equal	
Increase horizontal spacing	
Decrease horizontal spacing	
Remove horizontal spacing	
Make vertical spacing equal	
Increase vertical spacing	
Decrease vertical spacing	
Remove vertical spacing	

5. Choose File > Save.

Centering Controls in a Grid Layout

Grid layout allows you to center controls vertically or horizontally in the layout canvas.

To center controls

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Layout Canvas, hold down the Shift key, and then click the controls you want to center.

   The last item selected is the one to which all other items will be centered.
4. Perform the following tasks as necessary:

Task	Button
Center controls vertically	
Center controls horizontally	

5. Choose File > Save.

Aligning Label Text in a Grid Layout

You can align label text independently from its corresponding control.

To align label text

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. In the Layout Canvas, press Shift, and then click the labels whose text you want to align.

   If any of the items you selected is not a label, the text alignment buttons on the toolbar will be disabled.
4. Perform the following tasks as necessary:

Task	Button
Align left	

Task	Button
Align center	
Align right	

5 Choose File > Save.

Copying and Pasting Items in a Grid Layout

Grid layout allows you to copy and paste items.

To copy and paste controls

1 Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.

2 In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.

3 In the Layout Canvas, hold down the Shift key, and then select the items you want to copy.

4 Choose Edit > Copy.

5 Go to the Web template where you want the items to appear and then choose Edit > Paste.

- If you are pasting into the same applet Web template, new Control and Applet Web Template Item objects are created.
- If you are pasting into the same applet, but to another mode, only Applet Web Template Items are created.
- If you are pasting to a different applet, new Control and Applet Web Template Item objects are created.

When you choose Edit > Undo or delete Applet Web Template Items from layouts, controls are not automatically deleted. If you want to delete the controls, you need to do it manually.

CAUTION: If you are working in language override mode and you choose Edit > Undo after you have cut items from the applet layout, close the Applet Layout Editor without saving your changes. Otherwise. you will encounter errors.

Setting Tab Order for Fields in a Grid Layout

Grid layout allows you to set the sequence of fields that are activated each time a user presses the tab button. You can set the tab order by pointing and clicking in the Web Applet Layout Editor.

To set the tab order

1. Navigate to the applet you want to edit, right-click, and then choose Edit Web Layout.
2. In the Mode field of Web Controls toolbar, choose the applet mode that you want to edit.
3. Select the Format menu, and then select Set Tab Order.

   The Applet Layout Editor changes to Set Tab Order mode. Applet controls appear with a number next to them. The number indicates the sequence in which the user will progress through the controls using the tab button.
4. Define the tab order by clicking on controls in the sequence you want.

   Each time you select a control the control is assigned a sequence number.

   NOTE: If you do not click on a control, for example if you click on the layout background, the Web Applet Editor returns to normal edit mode and you must set tab order again from the beginning.
5. Choose File > Save.
6. Select Format > Set Tab Order.

   The Applet Layout Editor returns to normal edit mode.
7. Repeat steps Step 1 through Step 6 for each applet Web template mode.

Resizing the Grid Layout Canvas

You can resize the grid layout canvas.

To resize the grid layout canvas

1. In the Web Applet Editor, use the scroll bars to display the bottom right corner of the grid canvas.
2. Place your cursor over the canvas border (bottom edge, right edge, or bottom right corner) and drag it to the new position.

   The grid canvas is resized.

Converting Form Applets to a Grid Layout Using the Conversion Wizard

The Applet Web Template Conversion Wizard allows you to convert one or more existing applets to a grid-based layout. You may need to do this when you preserve custom layout during an upgrade (and have not previously converted to grid layout) and then want to convert form applets to grid layout. Or you may need to do this when you have custom applets that you have created using non-grid layout templates and then decide to convert them to grid-layout templates.

To convert one or more applets to a grid-based layout

1. In the Configuration Context Toolbar, make sure the application context that you want, such as All Applications, is selected.

   The wizard only converts controls that are valid in the current application context selected in the Configuration Context Toolbar. For example, if Siebel ERM is selected, only controls valid in the context of Siebel ERM are converted. If controls are not valid in the selected application context, a dialog box appears giving you the option of canceling the conversion or continuing. If you choose to continue, the controls not converted are written to a log file (`awtconversion.txt`) that is located in *`tools_install`*`\temp`. See "Troubleshooting Conversions to Grid Layout" on page 275.

2. If you are working in Language Override mode, make sure your Tools Language Mode is set to the language you want convert.

   For more information about Language Mode and Language Override, see *Using Siebel Tools*.

3. In the Object Explorer, select the Applet object type.

4. In the Object List Editor, select the applet or applets that you want to convert.

   You cannot convert applets based on the Web templates defined in "Applet Web Templates that Cannot Be Converted to a Grid Layout" on page 276.

   You cannot convert applets that do not have Web templates associated with them.

5. Do one of the following:

   - Right-click and then select Convert to Grid Layout.
   - Choose Tools > Convert to Grid Layout.

   The Applet Web Template Conversion Wizard appears.

6. Move the applets you want to convert from the Available Applets window to the Selected Window.

7. Select additional options.

   It is recommended that you select the Backup existing Applet Web Templates option.

   If you select the *Label on the left of the fields* option, the Conversion wizard creates a new non-grid form template, moves labels to the left, then converts that template to grid layout.

   If you select the *Launch web layout editor upon completion* option, the applet Web template displayed in the editor is the last applet that was selected in Step 6.

8 Click Next.

The wizard converts the active Web templates to grid-layout Web templates.

If no errors occur, you can edit the layout of these applets using the Web Applet Editor. For more information, see "Working with the Applet Layout Editor" on page 259.

If errors occur, they are displayed in a dialog box at the end of the conversion wizard. The same content is stored in a log file (`awtconversion.txt`) that is located in *`tools_install`*`\temp`. For more information about errors, see "Troubleshooting Conversions to Grid Layout" on page 275.

NOTE: Sometimes items in applet headers or footers are not converted properly and may have to be manually adjusted after the conversion. This can happen when a field is mapped to a placeholder in an applet header or footer. Typically controls mapped to headers and footers are buttons.

Converting Form Applets to Grid Layout By Changing the Web Template

You can also convert applets to a grid-based layout by changing the Web template associated with the applet. In Siebel Tools, you change the Web template file associated with each applet mode to a template that supports a grid-based layout. This is a manual task that you perform individually for each applet you want to convert.

There are two applet Web templates that support grid layout. See Table 42.

Table 42. Grid Layout Templates

Web Template	File Name	Comments
Applet Form Grid Layout	CCAppletFormGridLayout.swt	Use with all modes of form applets. This template has buttons in the applet header.
Applet Popup Form Grid Layout	CCAppletPopupFormGridLayout.swt	Use with all modes of popup form applets. This template has buttons in the applet footer.

See "About Form Applet Templates (Grid-Based)" on page 497 for more information.

To convert an applet to grid-based layout by changing the Web template

1 In the Object Explorer, select the Applet object type.

2 In the Object List Editor, select the applet that you want to convert to a grid-based layout and then right-click and choose Edit Web Layout.

You cannot convert applets based on the Web templates defined in "Applet Web Templates that Cannot Be Converted to a Grid Layout" on page 276.

3 In the Web Controls toolbar, click the Change Template button.

If the Web Controls toolbar is not displayed, choose View > Toolbars > Web Controls to display it.

The Choose Template dialog box appears.

4 In the Choose Template dialog box, select one of the following templates:

- For form applets, select Applet Form Grid Layout.
- For pop-up form applets, select Applet Popup Form Grid Layout.

5 Repeat Step 4 for each applet mode.

After associating a grid layout template, you can edit the applet layout using the Applet Layout Editor.

For general instruction on editing the layout of applets, see "Working with the Applet Layout Editor" on page 259.

For information on editing applets using grid-based layout features, see "Working With Grid Layout" on page 266.

Troubleshooting Conversions to Grid Layout

The error messages listed in Table 43 may appear during the conversion. These errors appear in a dialog box at the end of the conversion process and are written to a log file (`awtconversion.txt`) that is located in *`tools_install`*`\temp`.

Table 43. Conversion to Grid Layout Error Messages and Solutions

Error Message	Cause	Solution
Controls or labels cannot be mapped.	May be due to an Applet Web Template Item not being explicitly mapped to a control on the original applet Web template. Each Web template item must have the Control property populated to appear on the new grid-based applet Web template.	Use Applet Layout Editor to map controls to the applet layout. See "Adding Existing Controls and List Columns to Applet Layouts" on page 261.

Table 43. Conversion to Grid Layout Error Messages and Solutions

Error Message	Cause	Solution
Applet cannot be converted.	Applet does not have a Web template associated with it. Applet class or associated Web templates are not supported for grid layout. See "Applet Web Templates that Cannot Be Converted to a Grid Layout" on page 276.	Associate a Web template to the applet, by selecting the applet, then choosing Edit Web Layout.
The Applet Web Template...is configured for more than one application context.	The wizard only converts controls that are valid in the application context selected in the Configuration Context Toolbar at the time of the conversion. For example, if Siebel ERM is selected, only controls valid in the context of Siebel ERM are converted. If controls are not valid in the selected application context, a dialog box appears giving you the option of canceling the conversion or continuing. If you choose to continue, the controls that are not converted are listed in a dialog box and are written to the error log file. Note that the Expression property of the Web Template Item object type determines the application context for a control.	Select the appropriate application in the Application field of the Configuration Context toolbar and run the Conversion Wizard again. If the configuration Context toolbar is not displayed, choose View >Toolbars > Configuration Context. For more information, see "Converting Form Applets to a Grid Layout Using the Conversion Wizard" on page 273 and "About Application-Specific Mappings" on page 277.

Applet Web Templates that Cannot Be Converted to a Grid Layout

The applets and Web templates listed in the Applet Web Template Conversion Wizard's configuration file cannot be converted to grid layout. The file name is `awtcvtcfg.txt` and the file location is *`Tools_Install`*`\BIN`.

NOTE: Editing the list of applet classes and applet Web template files defined in the Applet Web Template Conversion Wizard configuration file (awtcvtcfg.txt) is not supported.

At the time of publication, the following Web templates cannot be converted to grid layout:

- SWLS DetailApplet Template
- SWLS Edit Template

About Application-Specific Mappings

You can configure controls and list columns for specific applications. For example, for a given application, you can display or hide controls, or reposition them in the applet layout.

The application setting of the Application drop-down list in the Configuration Context toolbar determines the setting applied to control mappings that are subsequently added or removed. By default, the layout editor is in All Applications mode, which leaves the controls that are added or deleted during the session unmodified. A specific application name can instead be chosen from the combo box, which places the layout editor in single-application mode, with the effect that controls that are added, deleted, or moved have that effect only for the selected application.

A conditional application-specific setting for a control is implemented with the Expression property in the Applet Web Template Item object definition for the control. The Expression property functions as a search specification or query condition, limiting the display of the control to those applications that match the expression condition. The Expression property is normally blank, which means "unrestricted," that is, the control appears in all applications. A single application name in the property value, such as eSales, restricts the control to appearing only in the specified application. A negation expression, such as NOT eSales, specifies that the control does not appear in eSales.

Do not set the Expression value directly. Instead, the value for this property is set by selecting a value from the Application drop-down list in the toolbar and then modifying the applet in the Applet Layout Editor. If you add a control to an applet and have an application selected, the application name is automatically entered in the Expression property for the control and the control only appears for the selected application.

NOTE: This feature could be used to add a new button that is only required in a particular application.

If you choose a different selection in the Application combo box during an editing session, the layout window changes to reflect the set of controls that is specified for that application. Namely, the controls appearing would be those specified for All Applications, plus those specified for the current application, less those negated for this application.

Unlike the target browser specific mappings, wizards do not affect application-specific mappings. If you use wizards to create something, it always gets created for all applications.

Configuring Controls and List Columns to Appear in More Mode Only

You can configure applets to show a limited set of fields by default and then show more fields when the user clicks the More button. The applet has two modes, a Less mode and a More mode. The user can toggle between these modes to display more or fewer controls or list columns.

The Mode property of the Applet Web Template Item object determines the mode in which the control or list column is displayed.

To configure controls or list columns to appear in More mode only

1. In the Applet Layout Editor, highlight the control.
2. Right-click and then choose More.

   A border appears around the control or list column and a down arrow icon appears next to it.

   The Mode property of the corresponding Web Template Item is set to *More*.

NOTE: The More/Less feature is not supported for Pop-up applets. For more information about Pop-up applets, see "Pop-Up Windows" on page 459.

13 Configuring Screens and Views

Topics in This Chapter

About the UI Navigation Model

Users navigate between screens and views using tabs, links, and drop-down lists that appear in one of the following four levels of the UI:

- **First level.** Screen tabs that allow users to navigate between screens.

- **Second level.** Links to groups of views (categories) or single views. These links can appear:
  - In the link bar directly below the screen tabs.
  - In a view drop-down list that appears in the header of applets. Typically, views that appear here filter data based on visibility rules. Examples of these types of views are My Contacts, My Team's Contacts, and All Contacts.
- **Third level.** View tabs that allow users to navigate to groups of detail views (detail categories) or single detail views.
- **Fourth level.** One of the following, depending on the Web template being used:
  - Links in the link bar directly below the view tabs.
  - View tabs on a grand-child applet.
  - Links in a drop-down list.

NOTE: There are other UI elements that also allow users to navigate, such as site map, drilldowns, and the thread bar.

Figure 62 shows the UI elements in each navigation level.

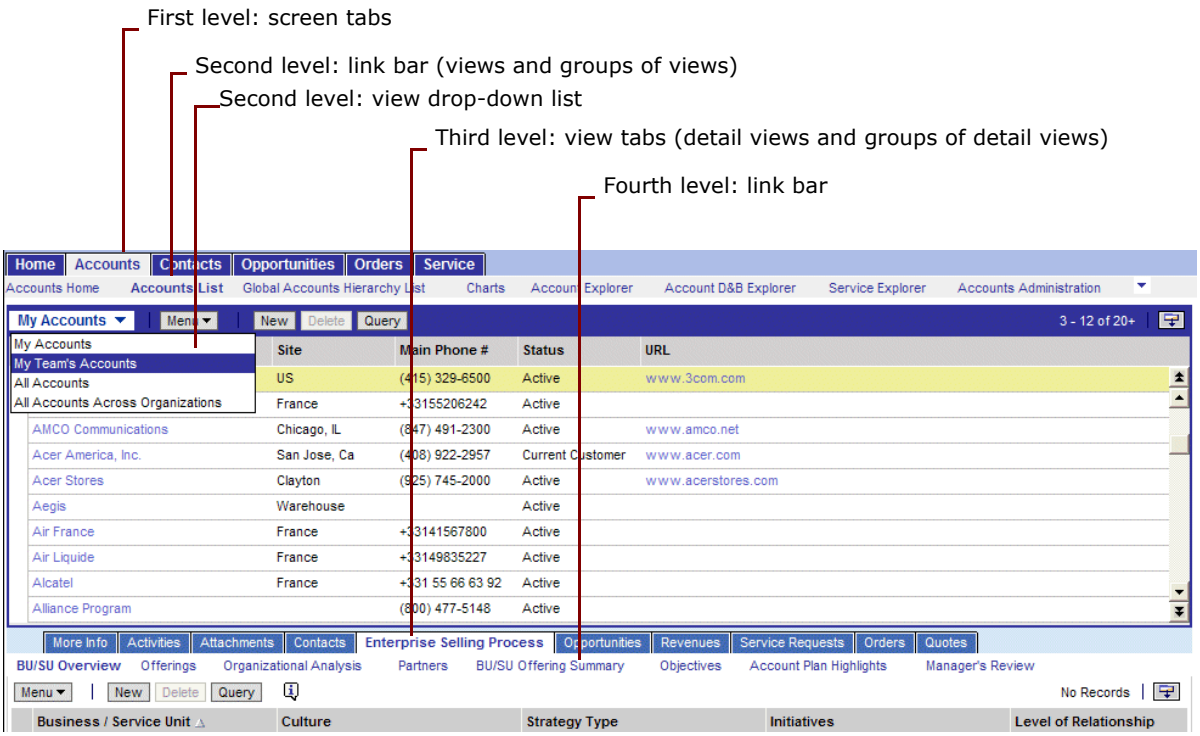


Figure 62. Navigation Levels

NOTE: The view shown in Figure 62 is a mock-up to demonstrate the different levels of navigation. This view does not appear in the Siebel repository.

Related Topics

"About Screen Views" on page 300
"Example Screen View Hierarchy" on page 303
"Process of Creating Screens and Screen Views" on page 304
"Configuring the Thread Bar" on page 292
"About Drilldowns" on page 294
"Defining Screen Menu Items" on page 322

About Views

A *view* is a repository object used to render the user interface. It is a collection of applets that presents data from a single business object. Views can contain lists, forms, charts, and other types of applets. Most views are either a master-detail view or a list-form view.

List-Form Views

A *list-form view* consists of a list applet and a form applet that display data from the same business component. The list applet appears above the form applet. It displays a list of records. The form applet presents detailed information about the record currently selected in the list applet.

Figure 63 shows an example list-form view.

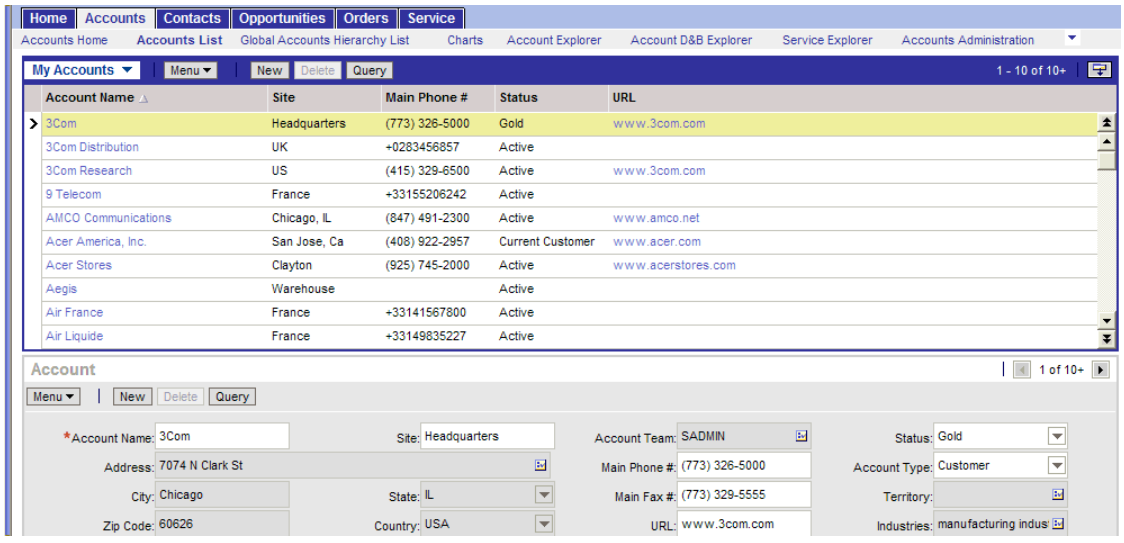


Figure 63. List-Form View

The applets shown in Figure 63 are the Account List applet and Account Entry applet. Both are based on the Account business component. The Account list applet displays a list of account records and the form applet displays detailed information about the selected account, but in a form that can be viewed without scrolling.

Master-Detail Views

A *master-detail view* typically consists of a form applet and a list applet that display data from two different business components. A link defines a master-detail relationship between the two business components. The form applet appears above the list applet and displays one record from the master business component. The list applet displays all of the records from the detail business component that are associated with the record in selected in the form applet.

NOTE: In another variant of the master-detail view style, the view can consist of two list applets. The records in the detail list applet are detail records of the currently selected record in the master list applet.

Figure 65 shows an example master-detail view. The applets shown are the Account Contact List Applet and the Account Entry Applet. They are based on the Contact and Account business components, respectively. The business object associated with the view is Account. In the context of the Account business object, the master-detail relationship between Account and Contact is based on the Account/Contact link.

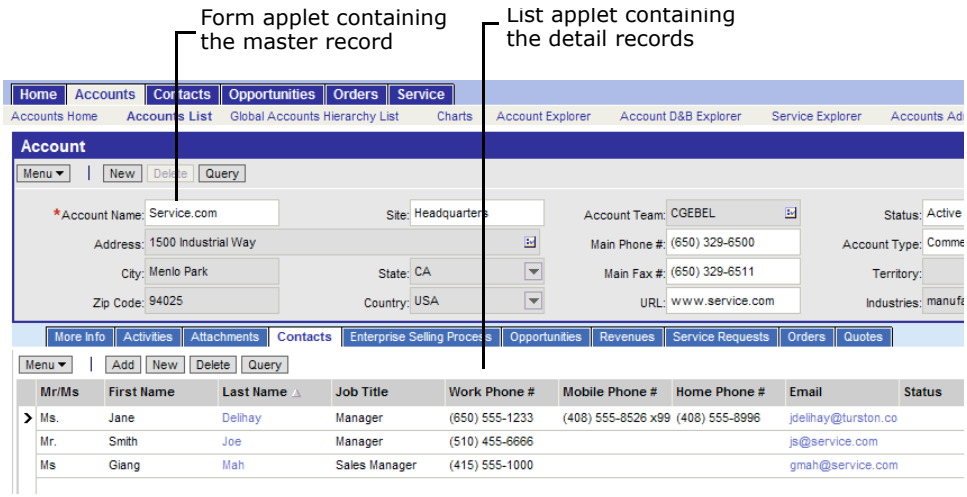


Figure 64. Master-Detail View

See Figure 65 for the relationships among the objects used to implement a master-detail view.

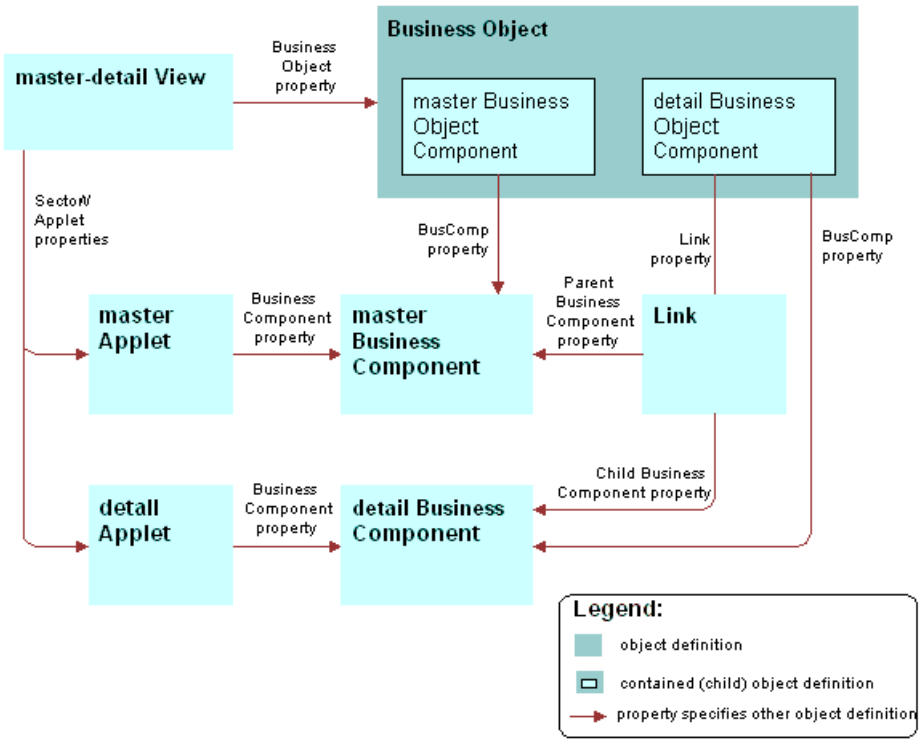


Figure 65. Master-Detail View Architecture

The object definitions shown in Figure 65 are briefly described below:

- **Master-Detail View.** The view being implemented.
- **Master Applet.** The form applet used to display the master record.
- **Detail Applet.** The list applet used to display the corresponding detail records.
- **Business Object.** Business object associated with the view by means of the Business Object property of the View object. The business object establishes the context that determines the active link between the business components associated with the two applets.
- **Business Object Components.** The business object components are child objects of the business object. Each business object component associates a business component to the business object.
- **Master Business Component.** The business component associated with the master applet.
- **Detail Business Component.** The business component associated with the detail applet.
- **Link.** The link that specifies the master-detail relationship between the master and detail business components. It is identified in the Link property of the detail Business Object Component.

Related Topics

"About Business Components" on page 167

"About Business Objects" on page 223

"About Links" on page 206

Guidelines for Configuring Views

When configuring views consider the guidelines in the following:

- Review the guidelines for configuring access control in *Security Guide for Siebel eBusiness Applications*.
- Do not associate views with more than one screen because it will cause the Thread Manager to behave incorrectly. When a thread is saved in the session file, the name of the view is stored without the name of the associated screen. When a user chooses a thread that navigates to a duplicated view, the user always navigates to one screen only, even if the thread was created in the other screen. Additionally, if the duplicate view is defined as the default view on both screen tabs, the user sees an anomaly in the user interface. One screen tab is selected as the active tab when either of the screen tabs are selected. The duplicate screen tab never appears to be active.
- Do not modify Administration - Server Configuration and Administration - Server Management views. Information in these views is read from the siebns.dat file and displayed in the user interface by the Server Manager. Configurations made to these views would also have to be made to the siebns.dat file. However, it is not possible to configure the product to store such information in siebns.dat. Therefore, configuration of server views is not recommended or supported.
- Due to the specialized nature of the code upon which the Siebel calendar is based, configuration of the following views is not supported:
  - High Interactivity Client Calendar View
  - eCalendar Daily View
  - eCalendar Weekly View
  - eCalendar Monthly View

  NOTE: Configuration of the eCalendar Detail View is supported through Siebel Tools.

Naming Conventions for Views

The following are general recommendations for view naming:

- Name a new view using a prefix that identifies your company. For example, a new view created for ABC Incorporated could be named ABC Opportunity Detail - Tasks View.
- View names should be meaningful. Avoid naming new views by adding a number suffix to an existing name (for example, Opportunity List View 2). If the view differs because it is read-only, then indicate this in your view name (for example, ABC Opportunity List View - Read Only).
- Initial-capitalize view names, for example, Opportunity List View rather than opportunity list view.

■ Do not use special characters such as an ampersand (&) when naming the views.

In addition, note the conventions in Table 44 for specific view types.

Table 44. Naming Conventions for Views

Type of View	Name Format	Example
List-form view	*buscomp* List View	Account List View
Master-detail view	*buscomp1* Detail - *buscomp2* View	Opportunity Detail - Contacts View
Explorer view	*buscomp* Explorer View	Account Explorer View
Chart view	*buscomp* Chart View - *Xxx* Analysis	Account Chart View - State Analysis

Typically, you should conform to a standard naming convention when naming views. Table 45 provides examples for each view visibility type. The *italicized text* identifies examples of the conventions you should use for each type of view visibility when naming views.

Table 45. Naming Conventions for View Visibility Types

View Visibility Type	Naming Convention
Personal	*My Personal* Contacts
Sales Rep	*My* Contacts
Manager	*My Team's* Contacts
Organization	*All* Contacts
Sub Organizations	*All* Accounts *Across My Organizations*
All	*All* Contacts *Across Organizations*
Group	*User Catalog* List View
Catalog	Products *Across Catalogs*
Admin Mode	Contacts *Administration*

Process of Creating Views

You typically create new views to expose new business components, business objects, or applets. The New View Wizard walks you through the steps of creating a view. After you create the view you must register it using administration screens in the Siebel application.

To create new views, perform these tasks:

1 "Creating Views Using the View Wizard" on page 286

2 "Registering Views" on page 287

Creating Views Using the View Wizard

The View Wizard steps you through the process of creating views. Use the View Wizard to create views because it prompts you for required information, sets property values, and configures dependent objects.

NOTE: The View Wizard can not be used when the ClientConfigurationMode CFG parameter is set to All.

To create a view using the wizard

1. Choose File > New Object from the Siebel Tools main menu.

   The New Object Wizard dialog box appears.

2. Click the View icon, and then click OK.

3. In the New View page, do the following:

   - Select the project that you want the view to be part of.
   - Enter a unique name for the new view.
   - Select the Business object whose data the view will display.
   - Enter the title for the view.

4. In the View Web Layout - Select Template page, select the template you wish to use for your new view.

   For a complete list of View Web Templates, see *Siebel Developer's Reference*.

5. In the Web Layout - Applets page, select the applets that you want to appear in the Web layout and then click Next.

6. In the Finish page, review your selections, and then click Finish.

   The View Layout Editor appears, allowing you to edit the view layout if necessary. For more information, see "Editing View Layout" on page 288.

   TIP: If the applets do not display properly in the view layout editor, exit the editor and make sure that the Applet Mode property for each View Web Template Item is set to a valid value. You can do this by launching the pick applet for that property for each applet. When you launch the editor again, the applets are rendered properly.

7. To preview the view, right-click in the Web Layout Editor, and then choose Preview.

8. Save your new view by choosing File > Save.

Registering Views

After creating a new view, you need to register the view in the Siebel application. Registering a view adds a new record for the view in the Siebel database and associates a responsibility with the view. All users with that responsibility will see the new view next time they log in. In a development environment, developers often perform this task to test a new view. In a production environment, this is typically an application administration task.

NOTE: This is an important step to remember. If you define a view without providing users access to it, it will not appear in the Siebel Web client.

To add the view to the Siebel application

1. Log in to the Siebel application as a user with administrative responsibilities, such as SADMIN.
2. From the Navigate menu, choose Site Map > Administration - Application > Views.
3. In the View Name field select the drop-down arrow.

   The View picklist appears.
4. Select the name of the view and then click OK.

To associate the view to a responsibility

1. From the Navigate menu, choose Site Map > Administration - Application > Responsibilities.
2. Select the responsibility that you want to associate the view with.

   NOTE: You cannot modify the SADMIN responsibility that ships as seed data with Siebel applications.
3. In the Views list, enter a new record for the view.

Depending on the nature of the new view and the users who need access to it, you may also need to:

- Add new responsibilities.
- Add employees to the new responsibilities.
- Make views read-only for a given responsibility. The read-only feature allows an administrator to implement read-only views using the run time client without having to create objects in the Siebel repository.

For instructions on how to work with responsibilities and employees, see *Security Guide for Siebel eBusiness Applications*.

Creating Views using the Object List Editor

You can use the Object List Editor to manually create a new view object. However, it is recommended that you use the New View Wizard because the Wizard prompts you for all necessary information and configures dependent objects for you.

To define a view in the Object List Editor

1. From the Object Explorer, select View.

   The Object List Editor opens.

2. Choose Edit > New Record to add a new view object definition.
3. Set the following properties for the new view record:
   - **Name.** (Required.) The name of the view. All references to views are done through its name.
   - **Business Object.** (Required.) The name of the business object used by the view. This determines the relationship between business components on which member applets are based.

     All the sector properties must have a value specified (except explorer view).
   - **Screen Menu.** If TRUE, the view should be included in the Site Map.
   - **Title.** Text string used as the window title.
4. In the Object List Editor, select the record, and then right-click and choose Edit Web Layout.

   The Select Template dialog box appears.
5. Select a view template and click Next.

   The Applet dialog box appears.
6. Select the applets you want to appear on the view and then click Next.
7. Review your selections and then click Finish.

   For information about configuring visibility, see *Security Guide for Siebel eBusiness Applications*.

   For more information about the New View Wizard, see "Creating Views Using the View Wizard" on page 286.

Editing View Layout

You edit the Web layout of a view in the Web Layout Editor. The Web Layout Editor allows you to edit the mapping between applets in the view and placeholders in the template.

To use the Web Layout Editor to modify the layout of a view

1. Select a view in the Object List Editor.

2 Right-click, and then choose Edit Web Layout.

   If the view has a template associated with it, the Web Layout Editor appears. The Web Layout Editor renders the underlying view template with mapped and unmapped placeholders.

3 Add an applet to the Web layout of the view by dragging it from the Applets window and dropping it onto an applet placeholder in the template.

   The Applets window shows all applets based on business components in the business object of the view. When an applet is mapped to a placeholder, the applet is displayed in the position it would appear at run time.

4 Delete an applet from the layout by selecting it, and then clicking the DELETE key.

5 Preview the view design by right-clicking on the layout, and then choosing Preview.

   The preview simulates what the view would look like in the run time environment by removing unmapped placeholders. This preview is not intended to be an exact representation of the eventual HTML output; it is intended to give you a rough idea about the structure and look of the generated output.

6 Export the preview to an HTML file by choosing File > Export, and then choosing a file name and location in the Save As dialog box that appears.

7 Change the Web template used by clicking the Change Template button that appears next to the Template text box in the Web Controls toolbar.

   NOTE: This might result in some mappings becoming invalid if the corresponding placeholder IDs do not exist in the new template. You can test for invalid mapping by right-clicking and selecting Check Mapping.

8 Save your changes by choosing File > Save.

Configuring Secure Views

You can create secure views for your Web application, using the HTTPS protocol. If a view is marked as secure, the Siebel Web Engine will verify that the current request used the HTTPS protocol, thereby preventing a client from obtaining access to a secure view by typing HTTP instead of HTTPS into their browser.

To specify that a view is secure

1 Navigate to a View.

2 In the Object List Editor, set the Secure property to TRUE.

   At run time, Siebel Web Engine will specify HTTPS protocol when it generates URLs to the view.

NOTE: The implementation of HTTPS is external to Siebel Web Engine. HTTPS is negotiated by the browser and the Web Server. Siebel Web Engine only specifies that HTTPS should be used for a particular view. Therefore, any server that is expected to provide secure views must have HTTPS enabled.

Configuring Views for Explicit Login

You can specify that users must type in their password and user name to access a view, if they have not already done so.

Users can log in to Siebel Web Engine applications using a cookie (after having selected Save My Username and Password), or by explicitly typing their username and password at the login page. If they have logged in using a cookie, you may still want them to supply their username and password to access a sensitive part of the Web site.

To specify that you want a view to require a login

1. Navigate to the View you want for which you want to require login credentials.
2. In the Object List Editor, set the Explicit Login property of the View to TRUE.

   When a user logs in using a cookie, and they attempt to access the view, they will be prompted to type their username and password. Users will only be required to do this once per session. After supplying their username and password, all subsequent visits to the explicit login view will not require login.

Enabling and Disabling High Interactivity for Views

You can control whether views appear in high interactivity mode by the underlying class definitions of the applets that appear on the view. A view is displayed in high interactivity mode when the underlying classes of the applets in the view have the high interactivity Enabled property set to 2, 3, 4, or 5. The possible values for the high interactivity Enabled property are summarized in Table 46.

Table 46. High Interactivity Enabled Property Values

Value	Works with High Interactivity	Works with Standard Interactivity	Cachable
1	No	Yes	No
2	Yes	No	Yes
3	Yes	No	No
4	Yes	Yes	Yes
5	Yes	Yes	No

For more information about standard and high interactivity, see "About Standard and High Interactivity" on page 37.

Troubleshooting View Configuration

There are several common mistakes that result in views not appearing in the application for a given user.

Possible reasons that a view is not appearing are:

- The view does not exist in the .srf file.

  This includes a possible misspelling when the view was registered (Site Map > Administration - Application > Views); that is, it does not match the view name in the .srf file. If it matches, compile the .srf file again using the All Projects option (full compile).

- The view is not included in one of the logged-in user's responsibilities.

  - Determine which responsibilities the logged-in user has (Site Map > User Administration > Employees).
  - Determine for each responsibility whether the view is included (Site Map > Application Administration > Responsibilities).

- The view is hidden using personalization rules.

  Determine this under Personalization Administration > Views. For testing purposes, you can also switch off the EnablePersonalization parameter in the .cfg file.

- The view is not included either in the menu or in the view tabs. In this case, the view can only be accessed by drilling down from another view.

  - In Siebel Tools, examine the Screen Menu property of the View object. It must be set to TRUE for the view to be included in the Site Map.
  - Determine whether the view is included in a screen and that the Viewbar Text property of the Screen View child object of the screen is set appropriately.
  - Determine whether the view's Visibility Applet and Visibility Applet Type properties are set correctly. For detailed information, see *Security Guide for Siebel eBusiness Applications*.

- The view belongs to a screen that is not included in the currently running application.

  - In Siebel Tools, determine whether the screen is included in the application (Screen Menu Item child object of the application).
  - Determine whether the application name is spelled correctly in the .cfg file.

- The view does not belong to the same business object as the screen's default view.

  Make sure that the view is based on the same business object.

- The view is not available due to upgrade problems.

  If an upgrade was done, make sure that it was successful by verifying all the log files that were created. The upgrade log files are found in the DBSERVER_ROOT\DB_PLATFORM directory.

- The view is not included in your license keys.

  If none of the previous reasons is responsible for the view not being visible, it is likely that the view is not included in your license keys. Send the license keys to Siebel Systems for examination. See also Alert 0041 on Siebel SupportWeb.

- The screen menu item or page tab is not translated into its target language.

  Make sure that for each screen associated with the application (Screen Menu Item object) there is a translated string available in the target language and a Screen Menu Item Locale child object. If not, the screen will not appear in the Site Map.

  Similarly, for a page tab to appear, the Page Tab object must have a translated string and a Page Tab Locale child object with the appropriate language code.

  For example, if the application runs in Norwegian, there must be Screen Menu Item Locale and Page Tab Locale objects with the Language Code property set to NOR.

Configuring the Thread Bar

The *thread bar* appears immediately below the application toolbar. It helps users keep track of their navigational path after drilling down on records that take the user to a view in a different business object.

The thread bar is shown in Figure 66.

Figure 66. Thread Bar

The following properties in each view object definition are set in order to configure thread behavior:

- **Thread Applet.** Specifies which of the applets appearing in the view supplies the data value for the thread field.
- **Thread Field.** The name of the field whose data value is included in the arrow box, following the Thread Title. This is a field in the business component associated with the applet identified in the Thread Applet property.
- **Thread Title.** The text used in the thread to identify the view. For example, in most of the views displaying Accounts (such as Account List view and Account Detail - Contacts view), the Thread Title is Acct.

Configuring Personal Layout Control

Certain views in Siebel applications, such as home page views, allow the user to control the layout of the view. For example, these views allow the user to:

- Reorder applets
- Collapse or expand applets
- Show or hide applets

For a description of this feature from the end-user perspective, see *Fundamentals*.

The user can edit view layout in two modes: Show mode and Edit Layout mode.

- **Show mode.** Allows users to edit applets using controls placed at the top of each applet. For example, home page views are displayed in Show mode until a user clicks the Edit Layout button.
- **Edit Layout mode**. Allows users more edit capabilities than the Show mode. This mode is presented to the user using a separate applet called the Layout Controls applet and appears after the user clicks the Edit Layout button.

To configure a view to support personal layout control

1. Set the User Layout property of the View's View Web Template object to TRUE.
2. Define the default layout of the view by setting the following properties of the View Web Template Items associated with the view.
   - **Display Size.** Determines whether the applet is minimized or maximized. Always Maximized indicates that the applet cannot be minimized by the end user.

     NOTE: Minimized and maximized are referred to as collapse and expand in the client user interface.
   - **Display Visibility.** Determines whether the applet is shown or hidden. Always Show indicates that the applet cannot be hidden by the end user.
   - **Move Range.** Defines a range in which the applet may be moved. For example, on an application home page with two columns, applets would specify a move range of either Column1 or Column2. Any applet with a move range of Column1 is movable only within the first (left) column. Any applet with a move range of Column2 is movable only within the second (right) column.

     If this property is not defined, the applet cannot be moved by the end user. The applet location is fixed within the view. For example, the salutation applet on the home page would typically not have move range specified for it.
3. Add the Layout Controls Applet to the view, add a corresponding View Web Template Item, and map the applet to a placeholder in the Web Template.

   There is an applet in the Siebel repository called Layout Controls applet. Add this applet to the view that you want to enable for personal layout control. It serves as a container for the controls that handle view-level operations, such as Reset Default Layout. In Show Mode, this applet appears as the Edit Layout button. In Edit Layout mode, which appears after the user clicks the Edit Layout button, this applet shows all applets on the view, and allows the user to Hide All Applets, Show All Applets, Reset Default Layout, and return to Show mode by clicking Done.
4. Add the following view layout controls to applets within personalized views, add corresponding Applet Web Template Items, and map the controls to the appropriate placeholders in the Web template.
   - ButtonMoveAppletUp
   - ButtonMoveAppletDown
   - ButtonHideApplet

- ButtonShowApplet
- ButtonMinimizeApplet
- ButtonMaximizeApplet

These view layout controls use Invoke Methods to manipulate the user's view layout preferences.

About Drilldowns

Drilldowns allow users to click a hyperlink in a field and be taken to another view that displays more information about the field. Drilldowns are used primarily in list applets. The drilldown object is a child object of applet. Drilldown behavior is not supported on MVG applets, pick applets, or association applets.

Drilldowns can be either static or dynamic. A static drilldown always takes the user to the same view. A dynamic drilldown takes the user to different views, depending on certain conditions, such as the value of a field.

Consider the following about drilldown behavior:

- If the driving applet of a view has a search specification, this search specification is also applied to the destination view when drilling down. For more information, see "Setting Applet Search Specifications" on page 253.
- If the target view of a drilldown object has a different visibility type from the original view, drilling down on a cell will take the user to the first record of the destination view and not to the drilldown record.

Static Drilldowns

Static Drilldowns take users to the same view under all circumstances. Figure 67 displays the property relationships between a list applet, business component, and view in a static drilldown configuration.

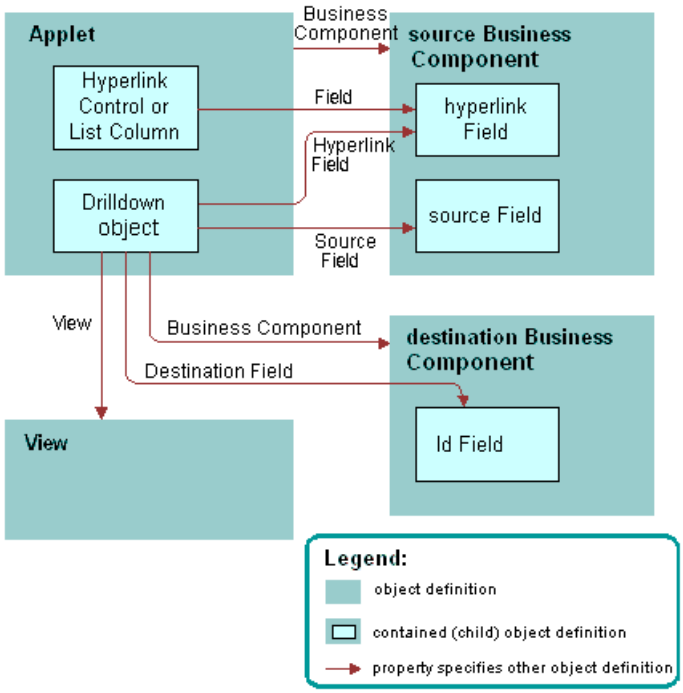


Figure 67. Static Drilldown Configuration

Dynamic Drilldowns

Dynamic drilldown enables hyperlink navigation to multiple views from the same hyperlink field, depending on the value of a field in the applet's current record.

This is useful in the situation where special processing is desired for various types of contacts, opportunities, accounts, and so on. The business component may have a field that indicates a classification, such as the Lead Quality for an opportunity or the primary Industry for an account. The drilldown behavior can be to check this field in the current record, and navigate to different views for different values found there.

Dynamic drilldown behavior for a hyperlink field (and the corresponding list column or control) is configured with one or more Dynamic Drilldown Destination child object definitions of the Drilldown Object. This is illustrated in Figure 68.

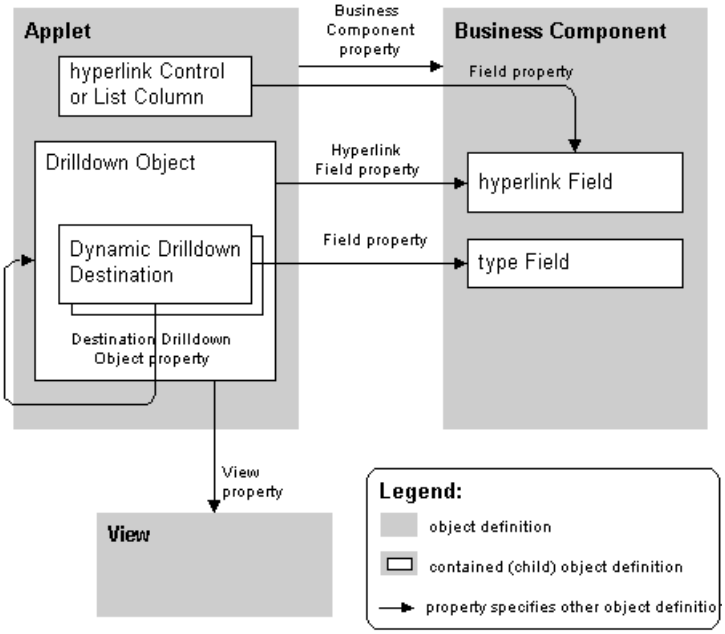


Figure 68. Dynamic Drilldown Configuration Details

As in a static drilldown configuration, the Drilldown Object object definition identifies a hyperlink field and a view. These property settings continue to have the same purpose in dynamic drilldown, namely to specify the list column or control that has hyperlink capabilities, and the destination view when the hyperlink is clicked.

However, for dynamic drilldowns, you define Drilldown objects for each candidate view. The Drilldown object with the *lowest* sequence number contains child Dynamic Drilldown Destination objects that are used to define the conditions under which each of the Drilldown objects should be activated. When the conditions defined in a Dynamic Drilldown Destination are matched, the logic routes to one of the candidate Drilldown objects. When all conditions expressed in the Dynamic Drilldown Destinations are false, the parent Drilldown object acts as the default.

For example, the Industry field in the Account business component could be designated as the type field in a list of Dynamic Drilldown Destinations. When the Industry value is "Manufacturing," the drilldown could route to a Drilldown Object with a view tailored for manufacturing accounts. When the value is "Transportation," the destination could be a different Drilldown Object and view, and so on.

The list of Dynamic Drilldown Destinations contained in a Drilldown Object specifies a set of criteria, of which any number may be met. If the condition in one Dynamic Drilldown Destination is met, the hyperlink routes to the specified Drilldown Object. If more than one is met, the first encountered (as specified in the Sequence property) specifies the destination Drilldown Object. If none is met (or no Dynamic Drilldown Destination object definitions are supplied as children of the Drilldown Object), the Drilldown Object itself supplies the name of the destination view.

Be careful to avoid routing hyperlinks from one dynamically evaluated Drilldown Object to another. That is, if you create Dynamic Drilldown Destination children of a Drilldown Object, do not have them route to a Drilldown Object that itself has Dynamic Drilldown Destination children. This practice could lead to ambiguity or looping.

If multiple drilldown objects for the applet are defined, a given field in the business component should be mentioned only once for all available drilldown objects. For a dynamic drilldown, the drilldown object that contains the dynamic drilldown destinations should have the Hyperlink Field property set.

About Applet Toggles

Applet toggles allow users to switch back and forth between different applets within the same view. This is useful when you want to display different types of data or present the same data in a different way. There are two types of applet toggles, static and dynamic:

- **Static applet toggles**. Allow users to toggle between applets by selecting the name of the applet from the Show drop-down list.
- **Dynamic applet toggles**. Automatically toggles between applets based on the value of a field in a parent applet.

When configuring applet toggles, consider the following:

- For applets involved in a toggle cycle the search spec on the form applet in the view will be applied first. Therefore, to apply a search specification on a list applet in a toggle cycle, you also need to add the search spec for the form applet.
- Dynamic toggle applets must be based on the same business component.
- Static toggle applets do not have to be based on the same business component.
- You cannot configure more than one applet toggle in a view.
- Applet toggles cannot be cached and therefore may have a negative affect on performance. Each time a user navigates to an applet, all available applet toggles are loaded. For more information, see *Performance Tuning Guide*.

Example of Configuring Applet Toggles

This topic gives one example of configuring an applet toggle. You may use this feature differently, depending on your business model.

Suppose you use the Contact business component to store information about your customer's preferred payment method. Payment methods are cash, credit card, or check. For each payment method, you need to gather different data:

- **Cash.** No special data requirement, this is the default payment method
- **Credit Card.** Credit Card Type, Credit Card Number, Expiration Date
- **Check.** Checking Account Number, Routing Number, Driver License Number, Driver License State

You could configure this example using a static toggle applet or a dynamic toggle applet. A static toggle applet would require the user to toggle between the different applets by selecting it from the Show drop-down list. Dynamic toggle would automatically toggle between applets based on the value entered in a Payment Type field.

To configure applet toggles (an example)

1. Create new fields in the Contact business component to capture the following information:
   - Payment Method (use a static bound picklist that contains Cash, Credit Card, and Check)
   - Credit Card Type
   - Credit Card Number
   - Expiry Date
   - Checking Account Number
   - Routing Number
   - Driver License Number
   - Driver License State
2. Expose the Payment Method Field in the Contact Form Applet. This is the default applet that would be used when the contact's preferred payment method is Cash.
3. Create two copies of the Contact Form Applet.

   Name one of them Contact Form Applet - Credit Card and the other Contact Form Applet - Check.
4. Expose the following fields in the Contact Form Applet - Credit Card:
   - Credit Card Type
   - Credit Card Number
   - Expiry Date BC Fields

   NOTE: When the contact's preferred payment method is Credit Card, the Contact Form Applet - Credit Card applet allows the user to enter a contact's credit card information.
5. Expose the following fields in the Contact Form Applet - Check:
   - Checking Account Number
   - Routing Number
   - Driver License Number

- Driver License
- State BC Fields

NOTE: When the contact's preferred payment method is Check, the Contact Form Applet - Check applet allows the user to enter a contact's checking account information.

6 Find and select the Contact Form Applet in the Object List Editor, navigate to the Applet Toggle child object, type and create the following records:

- **Record 1:**
  - Applet=Contact Form Applet - Check
  - Auto Toggle Field=Payment Method
  - Auto Toggle Value=Check
  - Name=Contact Form Applet - Check
  - Parent Name=Contact Form Applet
- **Record 2:**
  - Applet=Contact Form Applet - Credit Card
  - Auto Toggle Field=Payment Method
  - Auto Toggle Value=Credit Card
  - Name=Contact Form Applet - Credit Card
  - Parent Name=Contact Form Applet

NOTE: To configure this example as a static toggle applet that allows the user to select the appropriate applet from the Show drop-down list, leave the Auto Toggle Field and Auto Toggle Value properties blank.

7 Set Immediate Post Changes property of the Payment Method business component field to TRUE.

If the Immediate Post Changes property is set to FALSE, then the toggle does not occur until the user saves the record.

8 Compile your changes and test.

About Screens

A *screen* is a collection of related views:

- The screen represents a logical grouping of views pertaining to one business function.
- Views in a screen can be grouped into categories simplifying navigation for users.
- All the views in a screen usually map to a single business object.

Screens have a single child object type called Screen View. The screen view object type controls the views that appear in the user interface when a given tab is selected. See "Process of Creating Screens and Screen Views" on page 304.

You access screens through Screen Tabs or Site Map. The links for each of these are defined as part of the Page Tab object definition (child of screen)—one or both may exist for a given screen. Screen definitions specify the default view that appears when a tab is clicked.

NOTE: The Site Map is limited to nonvisibility views only. Visibility level views, such as My Accounts and My Team's Accounts, do not show up on the Site Map.

For more information about screen tabs, see "About the UI Navigation Model" on page 279.

About Screen Views

Screen view objects represent groups of views (categories) or single views in the user interface. They allow you to group related views together and control where the links appear in the user interface.

The Type and Parent Category properties determine where the screen view appears in the user interface. Screen view types are:

- **Aggregate Category.** Appears as a link in the link bar below screen tabs. Aggregate Categories are used to group all remaining screen view types.
- **Aggregate View.** Appears as follows:
  - When no value for the Parent Category property is defined, the screen view appears as a link in the link bar below screen tabs.
  - When the Parent Category property is set to a valid Aggregate Category, the screen view appears as a link in the view drop-down list in applet headers.
- **Detail Category.** Appears as a view tab. Detail Categories are used to group detail views.
- **Detail View.** Appears as follows:
  - When the Parent Category property is set to a valid Aggregate Category, the screen view appears as a view tab.
  - When the Parent Category property is set to a valid Detail Category, the view appears as a link in a link bar below view tabs, or in other locations depending on the Web template. For example, the screen view can also appear in a view drop-down list or in another row of view tabs.

For a visual description of the UI elements and navigation levels, see "About the UI Navigation Model" on page 279.

Table 47 lists each navigation level and UI placement and it summarizes the approach to configuring each level.

Table 47. Navigation Levels and Configuration Summary

Navigation Level	UI Placement	Object/Configuration
1	Screen tabs	Page Tabs (child object of Application) See "Defining Screen Menu Items" on page 322.
2	Links in the link bar directly below screen tabs (links to groups of views)	Screen View with the following properties: ■ Type = Aggregate Category ■ Parent Category = null See "Process of Creating Screens and Screen Views" on page 304.
	Links in the link bar directly below screen tabs (links to single views)	Screen View with the following properties: ■ Type = Aggregate View ■ Parent Category = null See "Process of Creating Screens and Screen Views" on page 304.
	Links in the view drop-down list in applet headers	Screen View object with the following properties: ■ Type = Aggregate View ■ Parent Category = an Aggregate Category See "Process of Creating Screens and Screen Views" on page 304.

Table 47. Navigation Levels and Configuration Summary

Navigation Level	UI Placement	Object/Configuration
3	View tabs (links to groups of detail views)	Screen View object with the following properties: ■ Type = Detail Category ■ Parent Category = an Aggregate Category See "Process of Creating Screens and Screen Views" on page 304.
	View tabs (links to a single detail view)	Screen View objects with the following properties: ■ Type = Detail View ■ Parent Category = an Aggregate Category See "Process of Creating Screens and Screen Views" on page 304.
4	Links in the link bar below view tabs, or in alternate locations depending on the Web template.	Screen View object with the following properties: ■ Type = Detail View ■ Parent Category = Detail Category See "Process of Creating Screens and Screen Views" on page 304.

Example Screen View Hierarchy

To understand the hierarchy of screen views, consider the example of the Accounts Screen and some of its child Screen View objects in the Siebel Repository. Table 48 lists several Screen Views from the Account screen and the key properties that determine the UI location.

Table 48. Example Screen View Hierarchy

Nav. Level	UI Location	Type	Category Name	Category Default View	View	Parent Category
2	Link in link bar below screen tabs	Aggregate View			Account Screen HomePage View	
	Link in link bar below screen tabs	Aggregate Category	Account List	Account List View		
	Link in view drop-down list in applet header	Aggregate View			Account List View	Account List
3	View tabs	Detail View			Account Detail - Contacts View	Account List
	View tabs	Detail Category	ESP	ESP Account Plan Overview View		Account List
4	Link in link bar below view tabs	Detail View			ESP Account Plan Overview View	ESP

Although the Screen View object plays a major role in determining where views appear in the UI, view Web templates ultimately control their appearance. For example, most Web templates are designed to render the 4th-level navigation as links under the view tabs. However, some Web templates are designed to render them as links in a drop-down list.

For information on how to configure Screen Views, see "Creating Screen Views" on page 304.

Process of Creating Screens and Screen Views

Screens objects define groups of related views. Screen views (child object of screen) identify the views and groups of views (categories) that you want to associate to the screen. You create a screen view object for each group of views and each view you want to appear in a given screen.

To crate screens and screen views, perform the following tasks:

1. "Configuring Screens" on page 304.
2. "Process of Creating Screens and Screen Views" on page 304.
3. "Defining Sequence for Screen View Objects" on page 306.

NOTE: To expose screens in the user interface, you must define two child objects of the Application object type: Page Tab and Screen Menu Item.

Related Topics
"Process of Exposing Screens in the UI" on page 320.

Configuring Screens

You create new screens in the Object List Editor in Siebel Tools.

To define a screen

1. In the Object Explorer, select the Screen object type.
2. Choose Edit > New Record to add a new Screen object definition.
3. Set the following properties for the new Screen record:
   - **Name.** Name of the screen. The Name is used by other objects to reference the Screen.
   - **Default View.** View that will be used when the user clicks on a page tab for a Screen.

     NOTE: A view must be added to the screen before it can be specified as a default.

Creating Screen Views

Screen View objects represent groups of views (categories) or single views in the user interface. They define which views appear for a given screen and determine where the view appears in the user interface.

For more information about where screen views appear in the UI, see "About Screen Views" on page 300 and "About the UI Navigation Model" on page 279.

When creating screen views, consider the following:

- Plan your user navigation using categories to group views where appropriate.
- Always set the Category Default View, otherwise the first view to be listed will be determined alphabetically by the Siebel Web Engine.
- Use Screen View Sequence Editor to determine the sequence of views. Do not edit the Sequence property of the Screen View Object manually. See "Defining Sequence for Screen View Objects" on page 306.

This task is part of the "Process of Creating Screens and Screen Views" on page 304.

To define a screen view

1. From the Object Explorer, select the Screen object type. The Object List Editor opens.
2. Choose Edit > New Record to add a new screen object definition.

   When you create a new record, the type defaults to Detail View and the Category Name and Category Default View properties are read-only.
3. Define a value for the Type property, and then use the information in the following table to set the property values for the new Screen View object:

Property	Description
Category Default View	Defines the default view for a Screen View of type Aggregate Category or Detail Category. It is recommended you always define this property for these types. For types of Aggregate View or Detail View, this property is read-only.
Category Name	Name of the Aggregate or Detail Category. This property must be unique within each screen. This field is Read-Only when a Screen View is of type Aggregate View or Detail View.
Display in Page	Must be TRUE for the Screen View to appear in the page. For Screen Views of Type Aggregate Category or Detail Category, this property should always be set to TRUE. It will not be displayed automatically if no views in the category are displayed.
Display in Site Map	Must be TRUE for the Screen View to appear on Site Map.
Menu Text	Text that appears in the Site Map. You can use HTML tags in this property to modify the text style that is rendered on the Site Map. For an example, see "Changing Styles of Label Text" on page 257.
Name	Calculated field.
Parent Category	If the Type property is Aggregate Category, this property is blank, because there is no parent. If the Type property is Aggregate View, this property may or may not be null. If the Type property is Detail View or Detail Category, this property must contain a value.

Property	Description
Type	Can be one of the following values: Aggregate Category, Aggregate View, Detail Category, or Detail View. See "About Screen Views" on page 300 for more information on Screen View Types.
View	Required when the Screen View is of type Aggregate View or Detail View. This property associates the View with the Screen View object. It is read only for Aggregate Category and Detail Category.
Viewbar Text	Text that appears in the user interface.

For more information, see "About the UI Navigation Model" on page 279.

Defining Sequence for Screen View Objects

You define the sequence in which views and groups of views (categories) appear in the user interface using the Screen View Sequence Editor. This is a visual design tool that allows you to see the hierarchy of screen view records at each level of navigation. It displays the hierarchy in a tree format and allows you to move individual screen views or groups of screen views to different positions in the sequence. However, you cannot move an item out of its current category or hierarchical level. If you do not define the sequence of screen views, the Siebel Web Engine will order them alphabetically.

The Sequence property displays a numerical code to indicate the sequence of a Screen View in the hierarchy. This field is updated when you open the Screen View Editor, make changes, and then choose File > Save. In the Object List Editor, you can sort on the Sequence column to see the hierarchy that will be displayed at run time. Note that if there is a null value for the Sequence property, SWE will render that Screen View as the last item in the sequence.

NOTE: Screen View sequence does not affect how views appear in Site Map. Views appear in Site Map in alphabetical order below the screen name.

To define the display order of Screen View objects

1. Navigate to the Screen that contains the screen views for which you want to define display order and select it.
2. Right-click and choose Edit Screen View Sequence.

   The Screen View Editor appears displaying the screen object and its child screen view objects in a tree structure. A screen view object displayed in bold indicates that it is the Category Defualt View for the category.

   You can also click anywhere in the screen view list, right-click, and then choose Edit Screen Sequence.

3 In the Screen View Editor, select a screen view object, and then right-click and use the options to move the screen view up or down in the tree:

Name	Description
Move to Next Higher Position	Moves the screen view one position up.
Move to Next Lower Position	Moves the screen view one position down.
Move to Highest Position	Moves the screen view to the highest position in its level of the hierarchy.
Move to Lowest Position	Moves the screen view to the lowest position in its level of hierarchy.

4 Choose File > Save and then exit the Screen View Editor.

Process for Creating Screen Home Page Views

Screen home page views provide users with a convenient way to access data and functionality for a given screen. Typically screen home page views contain applets for searching and adding records, as well as applets that display iHelp items, view links, and recent records. Screen home page views exist for screens such as Accounts, Contacts, Opportunities, Service, and Households. You can configure screen home page views for other screens as well. To configure a new screen home page view, complete the following tasks:

1. "Creating Rapid Search and Rapid Add Virtual Business Components" on page 308
2. "Configuring View Links Functionality" on page 311
3. "Configuring Recent Records Functionality" on page 312
4. "Creating a New Business Object for the Screen Home Page View" on page 313
5. "Creating New Screen Home Page Applets" on page 313
6. "Creating a New Screen Home Page View" on page 315
7. "Adding a New Screen View to the Screen Object" on page 317

Creating Rapid Search and Rapid Add Virtual Business Components

The Rapid Search and Rapid Add applets are based on virtual business components that reference the driving business component of a given business object. For example, the business components Account Home Search Virtual and the Account Home Add Virtual both reference the Account business component.

Using a virtual business component for each applet helps improve performance because when the screen home page view loads, an SQL query is not executed until the user submits a query or adds a record. It also allows both applets access to data from the referenced business component, while avoiding display problems that could occur if the two applets were based on the same non-virtual business component.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To create the Rapid Search Virtual business component

1 Create a virtual business component to represent the data presented by the target business component and use the following information to complete the record.

Property	Example Value	Comments
Name	Account Home Search Virtual	Follow the naming convention of *Target Business Component* Home Search Virtual *to* keep similar records in the repository consistent.
Class	CSSBCVMirrorAdd	Supports functionality of Rapid Add and Rapid Search with minimized system overhead to optimize performance.

NOTE: You must create virtual business components in the Object List Editor. You cannot use the Business Component New Object Wizard because it forces you to associate the business component with a table.

2 For the Home Search Virtual virtual business component, define child Field objects to represent the fields from the target business components that you want to appear in the search applet on the home screen.

The field names in the virtual business component must match the field names in the target business component.

CAUTION: MVGs are not supported on Rapid Search or Rapid Add applets.

3 For the Home Search Virtual business component, define the following business component user properties:

Name	Value	Example	Comment
Mirror Search Target BusComp	*Name of target business component*	Account	Required. Specifies the target business component.
Mirror Search Target BusObj	*Name of target business object*	Account	Required. Specifies the target business object.

To configure the Rapid Add virtual business component

1 Create a virtual business component to represent the data presented by the target business component and use the information in the following table to complete the record:

Property	Example Value	Comments
Name	Account Home Add Virtual	Follow the naming convention of *Target Business Component* Home Add Virtual *to* keep similar records in the repository consistent.
Class	CSSBCVMirrorAdd	Supports functionality of Rapid Add and Rapid Search with minimized system overhead to optimize performance.

NOTE: You must create virtual business components in the Object List Editor. You cannot use the Business Component New Object Wizard because it forces you to associate the business component with a table.

2 For the Home Add Virtual virtual business components, define child Field objects to represent the fields from the target business components that you want to appear in the search applet on the home page screen.

The field names in the virtual business component must match the field names in the target business component.

CAUTION: MVGs are not supported on Rapid Search or Rapid Add applets.

3 For the Home Add Virtual business component, define the following business component user properties:

Name	Example Name	Value	Example Value	Comment
Mirror Field *Field Name*	Mirror Field Account	Pick, *Target Field*, *Mirror Pick Id Field*	Pick, Account, Account Id	Optional. Used to display dynamic picklists (picklists that do not use LOVs). Indicates a pick field and identifies the field, the corresponding field in the target business component, and the base table Id field in the virtual business component.
Mirror Add *Mirror Pick Id Field Name*	Mirror Add Account Id	Ignored	Ignored	Optional. Used when Mirror Field *Field Name* defines a pick field. Prevents the Mirror Pick Id Field from being added to the target business component and causing record insertion failure.

For more information about configuring business components, see "Configuring Business Components" on page 167.

Configuring View Links Functionality

The View Links area on a screen home page displays links to the frequently accessed lists of data. Both administrators and users can define view links. For more information about defining view links, see *Fundamentals* and *Applications Administration Guide*.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To implement view links functionality on the screen home page

■ Add the following user property to the SRF Vlink Screen business component:

Name	Example Name	Value	Comment
VlinkScreen: *Screen Home Page Name*	VlinkScreen: Accounts Screen	Y	Required. Implements View Link functionality.

Configuring Recent Records Functionality

The Recent Records area on a screen home page view displays a list of the last five records in the current screen that a user has created, modified, or accessed (by drilling down on the record).

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To implement the Recent Records functionality

1 Navigate to the Recent Record business component and add the following business component user property:

Name	Example Name	Value	Example Value	Comment
Recent Record Track BC-*Screen Home Page Business Object Name*	Recent Record Track BC-Account Home	Name of the t*arget Business Component*	Account	Associates a tracked business component to a screen home page view through the business object.

2 Navigate to the target business component (for example, Account) and add the following business component user properties:

Name	Value	Comment
Recent Record Enabled	Y	Required to implement recent record tracking, which is disabled by default.
Recent Record Name Field	*Business Component field used to track recent records.*	Specifies the field used to track recent records. The field is displayed on the Recent Record applet on the screen home page view. Calculated fields are allowed.
Recent Record Type Field	*Type field to be used in dynamic drilldown*.	Optional. Specifies the type field to be tracked. It is not displayed on the Recent Record Applet. However, it can be used to define a dynamic drilldown so that users may be brought to a different view based on a given value.

Creating a New Business Object for the Screen Home Page View

Screen home page views are implemented using a separate, scaled-down copy of the business object to which the other views in a given screen map. For example, in the Accounts screen the Account Screen Home Page View is based on the Account Home business object. Other views in the screen are based on the Account business object.

NOTE: Because screen home page views are based on different a business object than the other views in a screen, you must associate iHelp items to both business objects. For example, to have the Create a New Account iHelp task appear on both the Activities screen home page view and the Activities screen views (My Activities, My Team's Activities, and so on), you must associate both the Action Home and the Action business objects with the iHelp item.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To create the Screen Home Page business object

1 Create a new business object using the information in the following table complete the record:

Property	Value	Example Value
Name	*Target Business Component* Home	Account Home
Project	ScreenHomePage	ScreenHomePage

2 Create child Business Object Component records for each of the following business components:

- *Target business component*; for example, *Account*
- *Target business component* Home Add Virtual; for example, Account Home Add Virtual
- *Target business component* Home Search Virtual; for example, Account Home Search Virtual
- Recent Record
- Public and Private View Links
- Salutation (eApps)
- Screen Home Task Assistant

For more information about configuring business objects, see "Configuring Business Objects" on page 223.

Creating New Screen Home Page Applets

Applets that appear in screen home page views are simplified versions of applets that appear in other views. Using simplified versions of standard applets helps keep screen home page views simple, easy to manage, and performing well.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To create screen home page applets

1 Create a banner applet by copying an existing banner applet, such as *Account Home Screen Homepage Banner* and change the following properties:

Property	Change to
Name	*Target Business Component* Home Screen Homepage Banner
Title	Name of the home screen. This value appears in the middle area of the home screen, above the list of view links.

2 Create a rapid search and rapid add applets by copying and existing applets, such as *Account Home Search Virtual Form Applet* and *Account Home Add Virtual Form Applet* respectively.

a Change the following applet properties:

Property	Change to
Name	*Target Business Component* Home Search Virtual Form Applet or *Target Business Component* Home Add Virtual Form Applet
Business Component	*Target Business Component* Home Search Virtual or *Target Business Component* Home Add Virtual These business component were created in "Creating Rapid Search and Rapid Add Virtual Business Components" on page 308 and "To configure the Rapid Add virtual business component" on page 310.

b Remove existing controls (child objects of Applet) that represent fields from the original business component and add new controls to represent the fields from the target business component, which are the fields you want to display in the search applet. Be sure that the controls map to fields defined in the business component. Controls not representing fields can be reused. You do not need to remove them.

c Remove existing Web Template Items (child objects of Applet Web Template Item) that represent controls from the original applet and add new web template items to represent controls from the new applet. Be sure to repeat this step for each Applet Web Template Mode. For example, you may need to add and remove web template items for both Base and Edit Mode.

For the Item Identifier property, use a number between 1300 and 1340. This range is available for Rapid Add and Rapid Search applets.

d Modify one of the following:

- If drilldown objects were defined on the source applet, modify the drilldown object in the new applet to identify a target view to go to after the user clicks the Go button. For more information about drilldown objects, see "About Drilldowns" on page 294.

- If either the *Mirror Add GotoView* or the *Mirror Search GotoView* Applet User Property were defined for the source applet, modify the value in the new applet to identify the target view to go to after the user clicks the Go button.

For more information about configuring applets, see "Configuring Applets" on page 231.

Creating a New Screen Home Page View

After all the underlying objects are created (business components, business objects, and applets), you can create a screen home page view to display the objects and data in the user interface.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To create a new screen home page view

1. Choose File > New Object > View.
2. Use the following information to complete the New View Wizard:

Property	Comment
Project	Choose ScreenHomePage.
Name	Follow the naming convention of *Target Business Component* Screen Homepage View. For example, *Account Screen Homepage View.*
Title	Follow the naming convention of *Target Business Component* Home. For example, Account Home
Business Object	Name of home page business object. For example, Account. See "Creating a New Business Object for the Screen Home Page View" on page 313.

Property	Comment
View Web Template	Choose View 25 50 25; this View Web template will provide a three column layout.
Applets	Select the applets you want to appear on the screen home page view. Typically, these include the following: ■ Layout Control SI Applet ■ *Target Business Component* Home Screen Homepage Banner ■ Public and Private View Link List Applet ■ Recent Record *Target Business Component* List Applet ■ Screen Home Task Assistant List Applet ■ *Target Business Component* Home Search Virtual Form Applet ■ *Target Business Component* Home Add Virtual Form Applet Rapid Search Virtual For more information about the Rapid Add and Rapid Search applets, see "Creating New Screen Home Page Applets" on page 313.

3 In the View Web Layout editor, verify the layout of the screen home page view.

The applet locations on screen home page views are described in the table below:

Applet	Location
Rapid Search	Left top
Rapid Add	Left bottom
Homepage Banner	Center To
Public and Private View Link	Center bottom
Edit Layout	Right top button
Screen Home Task Assistant List Applet	Right top
Recent Record	Right bottom

4 Navigate to the View Web Template Items for the view, and verify that the Applet Mode property is set correctly for each applet and verify the following Item Id properties for applet locations:

Applet	Applet Mode	Item Id
Rapid Search	Query	102
Rapid Add	Edit	103
Homepage Banner	Base	202
Public and Private View Link	Base	203
Edit Layout	Base	901
Screen Home Task Assistant List Applet	Base	302
Recent Record	Base	303

For detailed information about creating views, see "Process of Creating Views" on page 285.

Adding a New Screen View to the Screen Object

For the new screen home page view to appear in the application, you must create a new Screen View object to represent it.

This task is a step in "Process for Creating Screen Home Page Views" on page 308.

To create a screen view object to represent the screen home page view

1 Navigate to the Screen to which you are adding the home page view.

2 Define a new Screen View object using the following information:

Property	Value	Example
Name	*Name of screen view*	Accounts Screen Homepage View
Type	Aggregate View	
View	*Name of view*	Accounts Screen Homepage View
Viewbar Text	*Text to appear viewbar*	Accounts Home

3 Define the Screen View sequence by right-clicking in the Object List Editor and then choosing edit Screen View Sequence.

For detailed information about screen views, see "About Screen Views" on page 300.

Managing Unused Screens

If you will not use a standard screen in your implementation, there are two approaches to managing unused screens:

- You can use the Responsibility Administration Screen to disassociate all views of the unused screen from the responsibilities your organization uses. This approach does not require you to recompile the .srf file. It also offers an easy upgrade path if you decide to show the screen or views later. At that time, no configuration or software upgrade is required; you need only to reassign the views to the relevant responsibility.
- You can also inactivate the screen using Siebel Tools. This approach requires you to compile the .srf file. When you compile, the inactive screen will not be included in the .srf file.

14 Configuring Applications

Topics in This Chapter

About Applications

An *application* is a collection of screens. For a given application, the Application object defines which screens will be accessible through menus and tabs. Examples of applications are Siebel Call Center and Siebel Partner Relationship Manager.

Although you can create new applications, modifying an existing application is usually sufficient to meet customer business requirements.

Two child objects of Application associate Screens to the application and expose Screens in the user interface:

- **Page Tabs.** Add Screens to the Tab bar.
- **Screen Menu Items.** Add screens to the Site Map.

In addition to collecting a group of screens and their associated views, an application object definition also includes the following:

- Find Objects used to configure the Find dialog box. For more information, see "About Screens" on page 299.
- Server Script and Browser Script that can be implemented as event procedures on startup, prior to closing, and so on. These are implemented through Application Script child object definitions, and created and maintained in the Script Editor.

  For more information about Scripts, see *Siebel VB Language Reference*, *Siebel eScript Language Reference*, and *Siebel Object Interfaces Reference*.
- Custom menu options for Siebel-provided methods. These are implemented through the application method menu item object definitions, and created in the Applet Method Menu Item Wizard.

  For more information, see "About Applet Method Menu Item" on page 330.

Application Configuration Guidelines

When configuring applications, consider the following:

- Application names are case-sensitive and space-sensitive. A parameter in the configuration file(.cfg) for a given application identifies the Application Name.
- Application object definitions are contained in their own separate project. This is done to minimize locking of the application object definition.

Creating Applications

Usually modifying existing applications is sufficient to meet most customers' business requirements. However, there are cases when creating a new application is necessary.

For more information about when to reuse existing objects and when to create new objects, see "About Object Reuse" on page 51.

To create the Web application definition in Siebel Tools

1. Using the Object List Editor, create a new Application Object and enter the required properties such as Name and Project.
2. Add a page container template to the application for your home page.
3. Select a default login page, error page, and acknowledgement page from the list of available Web pages.
4. Associate the Web application with screens.

NOTE: You can also create a new application by copying an existing one and then modifying its properties and child objects. However, if there are application-specific fields on applets used by the original application, these fields (Applet Web Template Items) will not be available in the new, copied application because they are application specific, based on the application name.

Process of Exposing Screens in the UI

If you are adding new screens to an application, you need to perform the following tasks to expose Screens in the User Interface. Page tabs expose screens as tabs in the first level navigation. Screen Menu Items expose screens on the Site Map.

- "Defining Page Tabs" on page 321
- "Defining Screen Menu Items" on page 322

For more information about navigation levels, see "About the UI Navigation Model" on page 279.

Defining Page Tabs

Page Tabs are child objects of the Application object type. They appear as the first-level navigation in the user interface, allowing users to click the tab to go to the associated screen. Figure 69 shows Page Tabs in the user interface.

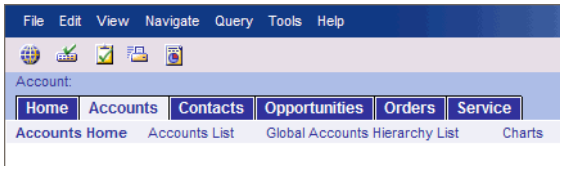


Figure 69. Page Tabs as Rendered in the User Interface

This task is part of the process "Process of Exposing Screens in the UI" on page 320.

To create Page Tabs in an application

1. In the Object Explorer, expand the Application object type.
2. Select the Page Tab object type.
3. Enter a new record, using the information in the following table:

Property	Description
Screen	The screen you want to expose through a page tab.
Sequence	Specifies the order of the page tabs for an application.
Text	Text string that appears screen tabs in the user interface. You can add HTML tags in this property to modify the text style that is rendered in the user interface. For page tabs, this works when running applications in standard interactivity mode. It does not work for applications running in high interactivity mode. For an example of using HTML tags to modify text stings, see "Changing Styles of Label Text" on page 257. **NOTE:** Translatable text string values are entered using symbolic string references or by entering values in a string override field. To add HTML tags to modify text strings, you must either create a new symbolic string that includes the HTML tags or enter the values in the string-override field. To be able to create symbolic strings or string override fields, you must have the EnableToolsConstrain parameter of the `tools.cfg` file set to FALSE. For more information about working with symbolic strings, see *Using Siebel Tools*.

Defining Screen Menu Items

Screen Menu Items are child objects of the Application Object Type. Screen Menu Items appear as hyperlinks on the Site Map. They allow users to click the hyperlink to go to the screen. Figure 70 shows Screen Menu Item definitions displayed on the Site Map.

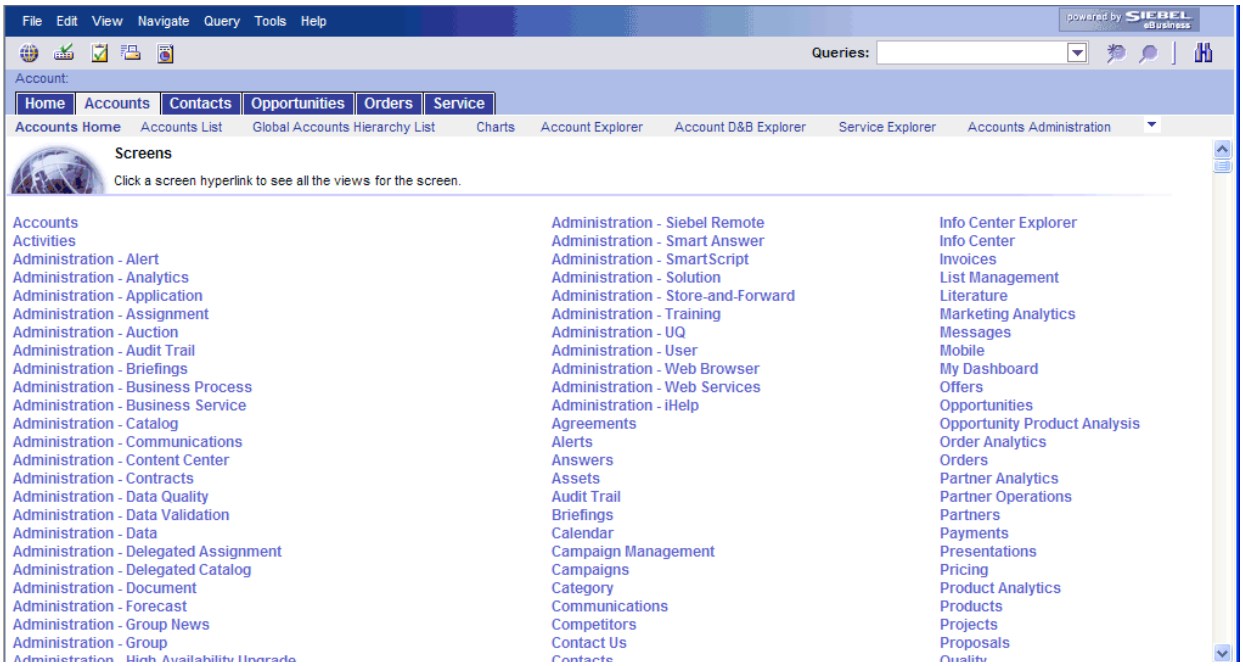


Figure 70. Site Map

Views that filter data based on visibility rules, such as My Accounts, My Team's Accounts, and so on, do not appear on Site Map. Screen Menu Items appear on the Site Map in alphabetical order.

NOTE: The site map is available from the application-level menu, by choosing View > Site Map.

This task is part of the process "Process of Exposing Screens in the UI" on page 320.

To associate screens with screen menu items

1. From the Object Explorer, expand the Application object type.
2. Select Screen Menu Item.

3 Choose Edit > New Record and use the information in the following table to define the object properties:

Property	Description
Screen	The screen that will be accessed with the menu item.
Text	Text string that appears in the site map in the user interface. You can add HTML tags in this property to modify the text style that is rendered in the user interface. For screen menu items, this works when running applications in standard interactivity mode or high interactivity mode. For an example of using HTML tags to modify text strings, see "Changing Styles of Label Text" on page 257. **NOTE:** Translatable text string values are entered using symbolic string references or by entering values in a string override field. To add HTML tags to modify text strings, you must either create a new symbolic string that includes the HTML tags or enter the values in the string-override field. To be able to create symbolic strings or string override fields, you must have the EnableToolsConstrain parameter of the `tools.cfg` file set to FALSE. For more information about working with symbolic strings, see *Using Siebel Tools*.

15 Configuring Web Page Objects

Topics in This Chapter

About the Web Page Object

The Web Page Object is the top-level object in the Web hierarchy that is used to create Web pages such as the following:

- Login pages
- Error pages
- Container pages

Editing the Layout of Web Page Objects

Like applets and views, Web pages are associated with templates. Web Page Items (child objects of Web pages) are mapped to placeholders in the templates. The Visual Web Page Editor allows a user to view and edit Web page objects.

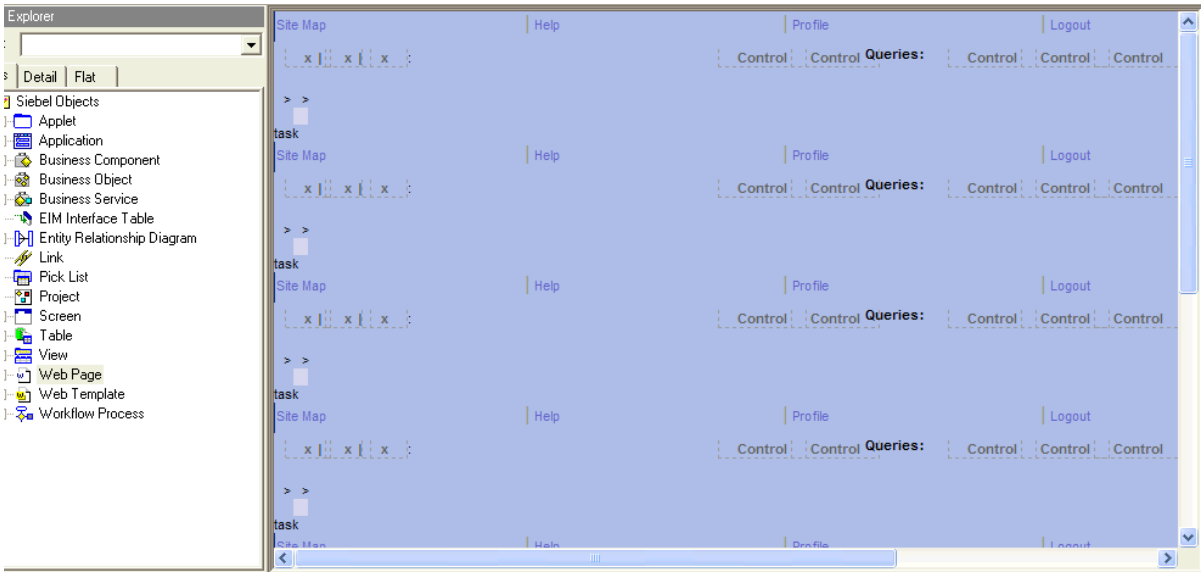
To access the Web Page Layout Editor

1. Select a Target Browser from the drop-down list on the toolbar.

   If you do not select a browser, an error message appears when you choose the Edit Web Layout option from the menu.

2 Navigate to the Web Page object type you want to modify, right-click, and then choose Edit Web Layout.

The Web Page - Layout window appears.



NOTE: The layout editor shows multiple images because the template on which the Web page is based contains a conditional tag such as<swe:if> and <swe:case>. The template content that is used varies depending on whether any one of the conditions is met. For example, in the case of the Page Container, the condition can simply be whether or not a CTI Java Applet is used or some other subtle or nonvisual differences. The effect is that the layout editor shows the page as if all the conditions were true. This is useful in case you want to edit any of them. However, typically only one condition would be true at run time, so you would not see the repeating images in the Web client.

3 Select a custom control from the combo box on the toolbar and drag it to a placeholder.

4 Set properties (like caption, invoke method, and so on) for the control using the Properties window.

If you click on the Web Page Item object type in the Object Explorer, the Web Page Items list applet displays these mappings.

16 Configuring Toolbars and Menus

Topics in This Chapter

About Toolbars and Menus

Toolbars and *menus* allow users to initiate various actions. The application-level menu (File, Edit, View, Navigate, Query, Tools, and Help) appears in its own frame near the top of the application in the browser window, and the application toolbar appears just beneath the primary tab bar, as shown in Figure 71.

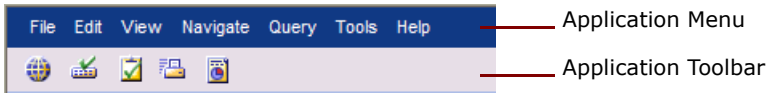


Figure 71. Toolbar and Application-Level Menu

Applet-level menus are located in the upper left corner of an applet. Each menu option lets you perform a task. An applet-level menu is shown in Figure 72.

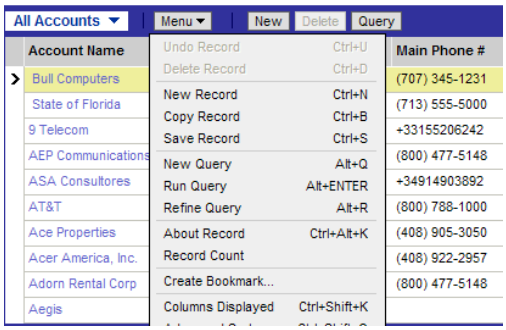


Figure 72. Applet Menu

The user's click on a toolbar icon or menu item is typically translated into a call to an invoke method, which may reside in a service on the browser or server, or in classes in the browser application or server infrastructure (applet or business component classes, SWE frame manager, or model). The toolbar icon or menu item is configured to target a method name, a method handler (from which it may be automatically retargeted if not found), and optionally a service.

Application-level items (which include both toolbar icons and application-level menus) are implemented through the use of Command object definitions in Tools, which are then mapped to Toolbar Item or Menu Item object definitions.

In Web templates, the `<swe:toolbar>` tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the `<swe:toolbaritem>` tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository.

For more information about Siebel Tags, see *Siebel Developer's Reference*.

The object types used to configure menus and toolbars are:

- Command
- Toolbar
- Toolbar Item
- Applet Method Menu Item
- Class Method Menu Item

About the Command Object Type

A Command object definition specifies which invoke method is called when a toolbar icon or application-level menu item associated with the command is executed (applet-level menus do not use command objects). It also specifies which bitmap appears on the toolbar icon for toolbar items. Command object definitions are referenced by Toolbar Item or Menu Item object definitions.

For more information about creating Command objects, see "Creating Command Objects" on page 334.

A Command object definition has the following significant properties:

- **Target.** Specifies which entity handles the invoke method the command calls. Available options are the following:
  - **Browser.** The method handler is a JavaScript service on the browser, or the JavaScript application, depending on whether a service is specified in the Business Service property.
  - **Server.** The method handler is an object manager service on the server or the object manager infrastructure (the SWE UDF loader, or, secondarily, the model), depending on whether a service is specified in the Business Service property.
  - **Browser Applet.** Used with high interactivity.

  For more details on the configuration of the Target property and related properties, see "About Invoke Method Targeting" on page 332.

- **Business Service.** Specifies the service (either browser or server, depending on the Target property) that handles the invoke method. If the property is left blank, the browser or server infrastructure is targeted rather than a specific service. If a service is specified, it must handle CanInvokeMethod and InvokeMethod for the method specified in the Method property.
- **Method.** Specifies the name of the method to invoke when the menu item or toolbar icon is selected. This is a required property.

  See "About Invoke Method Targeting" on page 332.
- **Method Argument.** Provides the means to pass an argument to the invoke method specified in the Method property. For example, a command item that opens a new window and navigates to a URL in that window can specify the GotoURL method in Method and the URL to navigate to in Method Argument.
- **Show Popup.** If TRUE, specifies that a new browser window is opened before invoking the method. If FALSE, specifies that the method is invoked in the current browser window.
- **HTML Popup Dimensions.** Dimensions, in pixels, of the pop-up window, when Show Popup is TRUE. An example is 640x480 (specified with the "x" and without spaces).
- **HTML Bitmap.** Specifies bitmap used by the Command object.
- **Tooltip Text.** This is the tooltip text which appears when the cursor lingers on a toolbar icon. For built-in methods, the tooltip text should be left blank; blank indicates that the method will dynamically supply the text, and language localization takes place as a part of this process. For developer-defined methods, you should enter literal text (but note that this turns off language localization for this tooltip text).

About the Toolbar Object Type

For each toolbar in the application, you create a Toolbar object definition in the Object List Editor. This provides a named toolbar that the user can activate or deactivate in Siebel applications, and to which icons (Toolbar Item object definitions) can be associated or removed. Typical toolbar functionality for most applications is implemented using HTML toolbars. In an HTML toolbar, the buttons are static images, which may be dimmed to indicate unavailability, but otherwise are not manipulated by program logic on the browser. In contrast, communications toolbars in applications such as Call Center, require toolbar icons that can be altered in response to events, such as blinking a particular toolbar icon when a call is incoming. This requires the use of Java toolbars. To specify that a toolbar is of the Java type, a class name is entered in the Class property.

For more information about configuring communications toolbars, see *Siebel Communications Server Administration Guide*.

Important properties of Toolbar are:

- **Class.** Left blank for an HTML toolbar, specified for a Java toolbar - the name of the Java class that implements the toolbar.
- **Name.** Referenced by other object definitions, and by the <swe:toolbar> tag in the "name=" clause.
- **Display Name.** Used for the History button and to show or hide toolbars by name.

About the Toolbar Item Object Type

The Toolbar Item object type associates a Command object definition (identified by name as a property in the Command property) with a Toolbar object definition (the parent of the Toolbar Item). This association places a toolbar icon, whose bitmap image, invoke method, and target are specified in the Command object definition, on the specified toolbar in a given location (relative to the other toolbar icons on that toolbar). The following properties are significant in a Toolbar Item object definition:

- **Command.** Name of the Command object definition that is to provide the bitmap, method, and target for the toolbar item. One or more hyphens can be specified instead of the name of a Command object to tell the system to insert a separator there between icons.
- **HTML Type.** Identifies the type of control to be displayed in the toolbar in the browser. Options include ComboBox, Button, Edit, Label, Hyperlink, MiniButton, and Timer.
- **Name.** Name of the toolbar item. Used internally in Siebel Tools only. This needs to be unique within the scope of a toolbar.
- **Position.** Sequence of the values determines the order of toolbar items. Also determines parent-child relationships. For example a Toolbar item with a Position value of 150 is a parent to Toolbar items with Position values of 150.10, 150.20, and so on. Parent Toolbar items must be active for child Toolbar Items to appear.

About the Menu and Menu Item Object Types

The Menu object type defines a named menu that is displayed in the user interface. You can add and remove menu items for each menu.

Menu Item object types associate a Command object definition with a Menu Item object definition. This association places a menu item whose invoked method is specified in the Command object definition on the specified menu in a given position. Significant properties are the following:

- **Name.** Uniquely identifies the menu or menu item.
- **Command.** Name of the Command object definition that is to provide the method and target for the menu item.
- **Caption.** The text displayed in the menu or menu item.
- **Position.** Sequence of the values determines the order of menu items. Also determines parent-child relationships. For example a menu item with a Position value of 150 is a parent to menu items with Position values of 150.10, 150.20, and so on. Parent menu items must be active for child menu items to appear.

About Applet Method Menu Item

Defines a menu item in the applet-level menu for the parent Applet object definition. Important properties are the following:

- **Menu Text.** The text displayed in the menu item.

- **Suppress Menu Item.** Default is FALSE. If TRUE, causes the class-level menu item of the specified name to be removed from the applet-level menu in the applet where this property is specified.
- **Command.** Name of the Command object definition that is to provide the bitmap, method, and target for the applet menu item.
- **Position.** The sequence of the menu item in the single-level list of menu items.

About the Class Method Menu Item

Class Method Menu Item is a child of Class. It adds (or suppresses) a menu item on applet-level menus for SWE applets of the specified applet class and its subclasses.

Significant properties are the following:

- **Target.** Specifies which entity handles the invoke method specified in the Method property. Available options are the following:
  - **Browser.** The method handler is a JavaScript service on the browser, or the JavaScript applet class (secondarily the JavaScript business component class) on the browser, depending on whether a service is specified in the Business Service property.
  - **Server.** The method handler is an object manager service on the server or the applet and business component and their superclasses, depending on whether a service is specified in the Business Service property.
- **Menu Text.** The text displayed in the menu item.
- **Method.** The method invoked when the item is selected.
- **Business Service.** If specified, identifies the service on which to invoke the method. If unspecified, the method is invoked on the applet class on the browser or server (as specified in the Target property) with subsequent retargeting if unhandled.
- **Suppress Menu Item.** Default is FALSE. If TRUE, causes the applet-level menu items of the specified name to be removed from the applet-level menu in all applets derived from this class and its subclasses.
- **Position**. The sequence of the menu item in the single-level list of menu items.

About Activating and Deactivating Menu Items and Toolbars

Menu items (both application-level and applet-level) and toolbar items can be activated or disabled at run time, by means of the CanInvokeMethod mechanism. CanInvokeMethod (for the method specified in the Command, Applet Method Menu Item, or Class Method Menu Item object) will be called automatically for each item prior to displaying the menu or toolbar. If CanInvokeMethod returns FALSE, the toolbar item or menu item is not displayed. The CanInvokeMethod logic in most cases is retargeted from the browser application to the applet class hierarchy on the server, and from there to the business component class hierarchy. The targeting sequence is described below in "About Invoke Method Targeting" on page 332.

Suppression and activation of individual applet-level menu items at design time is supported by means of the Suppress Menu Item property in the Class Method Menu Item and Applet Method Menu Item object types. This is applicable to applet-level menus only, not application-level menus or toolbars, in which the item must be added or removed explicitly in Tools. Design-time menu activation or suppression for applet-level menus provides the means to make a menu item available globally for applets of a given class and its subclasses, and then suppress it in particular applets where it is not desired. Certain applet-level menu items appear in virtually all applets (such as Copy, Edit, and Delete), others appear in virtually all list applets (such as Columns Displayed), and so on, but there are always exceptions in which a "standard" menu item for the applet's class needs to be suppressed for a specific applet.

To add applet-class-level menu items, you would add a Class Method Menu Item for a standard menu item for a given applet class. This menu item would not need to be re-included as Applet Method Menu Item object definitions in applets where you want the menu item to appear. You would only create Applet Method Menu Item object definitions in two circumstances: to add a menu item (not already provided by the applet's class) to the applet, or to suppress display of an applet-class-level item that the applet would normally inherit. In this latter case, you create an Applet Method Menu Item object definition with the same name as the applet-class-level menu item you want to suppress, and enter a value of FALSE for the Suppress Menu Item property.

About Invoke Method Targeting

The Method, Business Service, and Target properties appear in the Command object type for use in toolbars, application-level menus, and applet menus. The target property specifies the object or service that will process the method invoked by the command. Under some circumstances, if a method cannot be handled by the specified target it is automatically directed to an underlying object or service for handling. This could be a mirror instance of the object that exists on the server rather than the browser, or it could be an inherited class. In these cases we say that the method invocation has been retargeted.

Two settings are available for the Target property, with the following behavior:

- **Browser target.** The method handler for this target is the JavaScript application, a JavaScript applet, or a JavaScript service, on the browser side. In all cases, a method name must be specified in the Method property. A service is targeted if a service name is specified in the Service property. If a service is not specified, method handling differs based on whether the calling entity is application-level or applet-level, as follows:

- **Application-level.** Targets to the specified method in the JavaScript application. Does not retarget.
- **Applet-level.** Targets to the specified method in the JavaScript applet. If not handled, retargets to the specified method in the corresponding JavaScript business component. No inheritance or additional retargeting.

- **Server target.** This target is for invoking a method in a C++ class on the server, either on a service or on the infrastructure. If a Service property value is not specified, the invoke method is targeted to the infrastructure. It will target the infrastructure differently depending on whether the menu or toolbar icon invoking the method is applet-level (menu only) or application-level (menu or toolbar).
  - **Application-level.** The method handler is initially the SWE UDF loader on the server side, and secondarily the model.
  - **Applet-level.** The method handler is initially the applet class to which the applet belongs, and is retargeted successively up through the applet class hierarchy to CSSSWEFrame. If still unhandled, handling is retargeted to the business component class of the applet's business component, and successively upwards through the business component class hierarchy to CSSBusComp.

If a service is specified in the Service property, the method handler is the specified service. This targeting is also dependent on whether the calling menu item or toolbar icon is applet-level or application-level, as follows:

- **Application-level**. The method handler is the specified OM service. It does not retarget.
- **Applet-level.** The method handler performs a SetBC call to set to the business component of the applet, and then calls the specified OM service. It does not retarget.

The results of the possible settings of the Target and Business Service properties at the applet and application levels are summarized in Table 49.

Table 49. Target and Business Service Properties Matrix

Menu/ Toolbar Level	Target	Service	Result
Application level	Server	Specified	The method handler is the specified business service on the server. It does not retarget.
		Unspecified	The method handler is the base functionality associated with an application object.
	Browser	Specified	Targets to the method in the specified browser-side service. It does not retarget.
		Unspecified	Targets to the specified method in the JavaScript application. It does not retarget.

Table 49. Target and Business Service Properties Matrix

Menu/ Toolbar Level	Target	Service	Result
Applet level	Server	Specified	The method handler calls the specified service on the server. It does not retarget.
		Unspecified	The method handler is initially the applet class to which the applet belongs, and is retargeted successively up through the applet class hierarchy to CSSSWEFrame. If still unhandled, handling is retargeted to the business component class of the applet's business component, and successively upwards through the business component class hierarchy to CSSBusComp.
	Browser	Specified	Targets to the method in the specified browser-side service. It does not retarget.
		Unspecified	Targets to the specified method in the JavaScript applet. If not handled, retargets to the specified method in the corresponding JavaScript business component. There is no inheritance or additional retargeting.

Creating Command Objects

A command object definition specifies which invoke method is called when a toolbar icon, application-level menu item, or applet menu item is associated with the command is executed. It also specifies which bitmap appears on the toolbar icon for the toolbar items. Command object definitions are referenced by Toolbar Item, Menu Item, or Applet Method Menu Item object definitions.

NOTE: The target property for the command object, displays six values in the picklist. The only valid values are Browser and Server. These values are the only options when you use the Command Wizard to create a new command object.

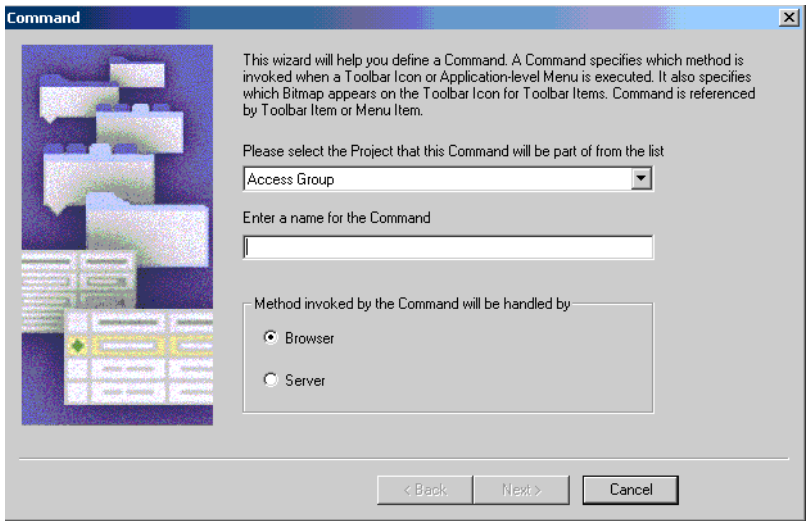
To create command objects using the Command Object wizard

1 Choose File > New Object.

The New Object dialog box appears.

2 Click the Command object icon.

The Command dialog box appears.



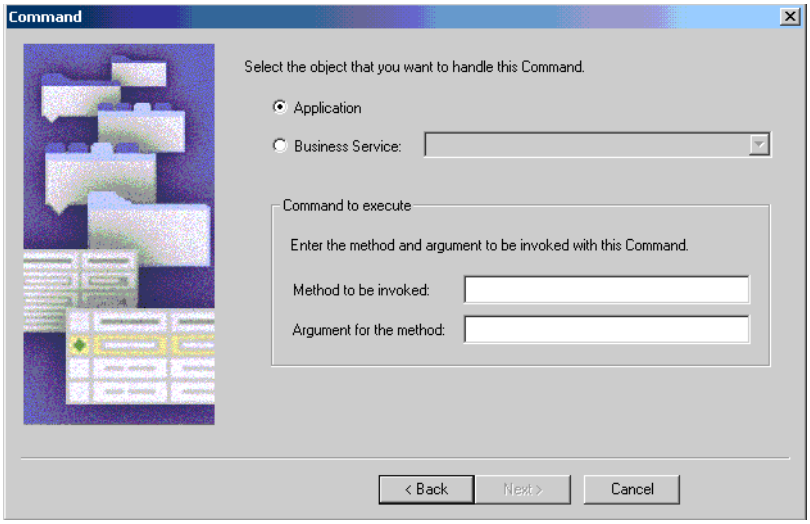
3 In the Command dialog box, do the following:

- Enter the project.
- Enter a unique name for the command object.
- Select whether you want the Method invoked by the command to be handled by the browser or the server.
- Click Next.

4 In the next dialog box, do the following:

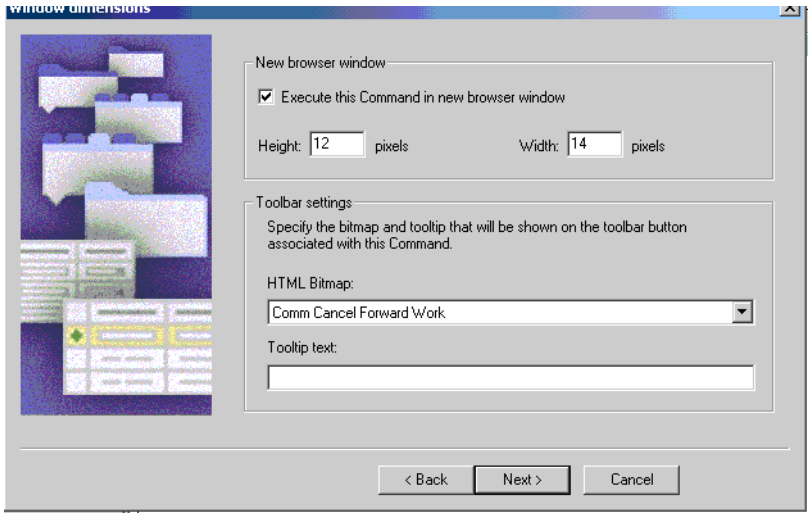
- Select the object that will handle the command. If the command is to be handled by a business service, specify the business service from the drop-down list. You must know whether the selected business service is available for your choice of browser/server.
- Enter the Method to be invoked by the command. Specify the method to be invoked by the command. You are responsible for choosing a method that is available for the business service or application chosen.
- You can provide the argument to be passed to the method (this is optional). The argument must be correct for the chosen method.

- Click Next.

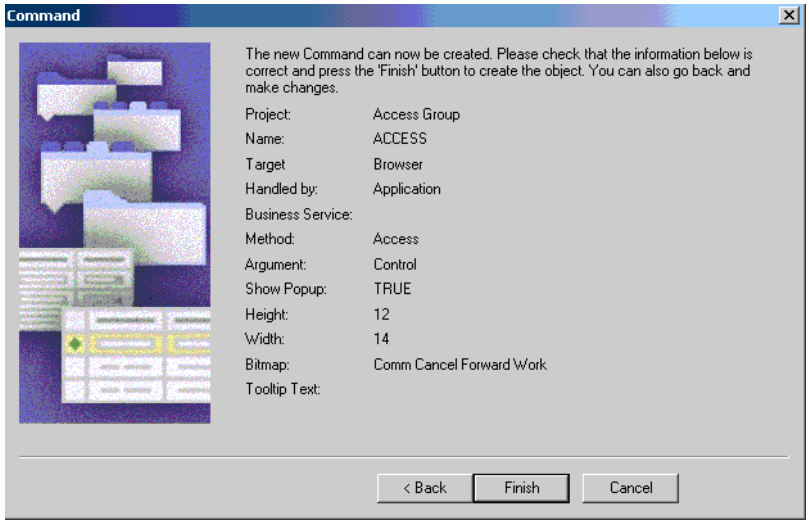


5 In the Window dimensions dialog box, do the following:

- Specify if the command should be executed in a new browser window. If it should be, the window's height and width must be specified. Both these values must be valid integers.
- Specify the HTML bitmap and the tooltip text to be shown on the toolbar button associated with the command (optional).
- Click Next.



6 In the Command dialog box that appears review your entries.



7 If any corrections need to be made, click Back to return to the appropriate page on which the correction is to be made.

If the properties are correct, click Finish to generate the command object.

Creating a New Toolbar

You can create new toolbars for an application by defining a toolbar object and modifying the appropriate Siebel Web templates to expose the new object in the user interface. You can also add new icons to existing toolbars.

To define a new toolbar

1 In Types view in the Object Explorer, double-click the Toolbar object type.

2 Click to the left of a row in the Object List Editor, and then choose Edit > New Record.

3 Specify the name of the new toolbar in the Name property of the new object definition.

4 To expose the toolbar to the user interface, you need to add a specific tag to the Container Page or one of its child templates that you are using.

For detailed information on templates and tags, see *Siebel Developer's Reference*.

Adding a New Toolbar Icon to an Existing Toolbar

To add a new toolbar icon to an existing toolbar

1. Verify that the bitmap image you want to use on the toolbar icon surface currently exists as a child bitmap object definition of the bitmap category object definition named Command Icons.

   If it does not exist, create a bitmap object definition in this bitmap category as described in "About Displaying Images in the Siebel User Interface" on page 475. If it does exist, note the name of the bitmap object definition.

2. Verify that the method you want for this toolbar icon to invoke currently exists, or add a Siebel VB or eScript script to the application PreInvokeMethod.

   You need to write an If or Case statement based on MethodName and write the instructions for that MethodName within the If or Case statement.

   You also need to change the last line of PreInvokeMethod to CancelOperation (from ContinueOperation).

3. Navigate to the Command object type in the Object Explorer, and add a new Command object definition in the Object List Editor. Specify the HTML bitmap to use in the Bitmap property, the method to invoke in the Method property, and other properties.

4. Navigate to the Toolbar object definition to which the new Toolbar Item is to be added.

5. In the Object Explorer, select the Toolbar Item object type.

6. In the Object List Editor, add a new Toolbar Item object definition. Specify the name, the name of the Command object definition that supplies the bitmap and method, and the position number of the toolbar icon relative to the other toolbar icons appearing in the Toolbar Items list in the Object List Editor.

Extending Toolbars Using JavaScript

You can take advantage of the functionality of high interactivity by extending JavaScript toolbars and creating new ones. Creating new JavaScript files to extend JavaScript toolbars is necessary if you need more toolbar icon types than the standard ones.

To extend a toolbar using JavaScript

1. Create a JavaScript file to define an extended JavaScript toolbar class that is a subclass of JSSToolbar.

2. Copy the JavaScript file to

   SIEBSRVR_ROOT\webmaster*<Siebel_build_number_in_use>*\scripts.

3 In Siebel Tools, create a DLL object as shown in the following table.

Field	Value
Name	User-defined name for the DLL object, for example BarcodeToolbar
Project	A currently locked project in the Siebel Repository
File Name	File name that references the JavaScript file, for example barcodeToolbar.js

4 Create a Class object as shown in the following table.

Field	Value
Name	Name of the class defined in the JavaScript file, for example JSSBarcodeToolbar
Project	The locked project used in Step 3
DLL	Name of the DLL object created in Step 3
High Interactivity Enabled	1

5 If you are creating a new toolbar, create a Toolbar object, as shown in "To define a new toolbar" on page 337.

The Class property must be the class defined in the JavaScript file, for example JSSBarcodeToolbar.

6 Add new toolbar items as shown in "To add a new toolbar icon to an existing toolbar" on page 338.

7 If you are creating a new toolbar, add a <swe: toolbar> tag to the appropriate Web template as shown in "How Toolbars are Displayed in Templates" on page 523.

The name property in the swe: toolbar tag must be the name of the Toolbar object in Step 5.

8 Add <swe: toolbaritem> tags to the appropriate swe: toolbar tag as shown in "How Toolbars are Displayed in Templates" on page 523.

Creating Applet Menus

You can configure the applet menus that come with Siebel applications and also create custom applet menus of your own using the Applet Method Menu Wizard.

The Applet Menu Wizard will allow you to modify an applet's method menu (applet level menus). Applet method menus are constructed by inheriting method menu items from the class to which the applet belongs and its super classes, and also by explicitly creating method menu items for the applet. Using the wizard, you can do the following:

- Suppress inherited method menu items
- Resurrect inherited method menu items
- Create new method menu items for an applet
- Delete existing method menu items of an applet

To use the Applet Method Menu Wizard

1. Choose File > New Object.

   The New Object dialog box appears.

2. Select the Applet Method Menu object.

   The Applet Method Menu dialog box appears.

3. In the Applet Method Menu dialog box, do the following:

   - Specify the applet whose method menu you wish to modify and also the project for the applet.

     Only projects locked by the user will appear in the drop-down list.

   - Click Next.

4. In the second Applet Method Menu dialog box, do one of the following:

   - Make a menu item visible in an applet by moving the item to the Selected Menu Items window.
   - Suppress a menu item by moving it out of the Selected Menu Items Window.
   - Click Finish or select Create New Menu Item. If you click Finish, all the changes that you have made are committed to the repository, and you are taken to the applet's object definition in the Object List Editor. If you select Create New Menu Item, the Finish button becomes the Next button.

5. If you are creating a new menu item, and you have checked the box labeled Create New Menu Item on the Applet Method Menu dialog box, then click Next.

6. Create a new method menu item definition by selecting an entry from the Select the Command to be executed by this Menu field.

7. From the Enter the text to be displayed for this Menu Item field, specify the text to display for this method menu item, and then click Next.

   The Applet Method Menu Item dialog box appears.

   You can examine the properties that you specified earlier. If any changes have to be made, you can click Back and return to the appropriate dialog box on which the correction is to be made.

8. If the properties are correct, you can click Create Menu Item to generate the method menu item object. After the item is generated, the Back button is deactivated and the Next button becomes enabled.

9 Click Next.

The second Applet Method Menu dialog box appears and the method menu item you just created is displayed in the Selected Items list box.

10 Click Finish.

The Applet Layout screen appears.

17 Configuring Picklists and Pick Applets

Topics in This Chapter

Types of Picklists

Picklists allow users to populate a field by selecting values from a list rather than typing them in fields. There are two types of picklists: static and dynamic.

- Static picklists draw their data from the Siebel list of values table, which is maintained by an administrator. The data in the list of values table is fairly static.
- Dynamic picklists draw their data from other user-maintained tables, such as S_CONTACT or S_ORG_EXT. The data of these tables is dynamic.

Related Topics

"About Static Picklists" on page 344

"About Dynamic Picklists" on page 352

About Static Picklists

A static picklist is a list of predefined values that the user invokes from a field in an applet. When the user clicks the drop-down arrow to the right of the field, a single-column picklist appears. The user selects a value from the list, and then clicks Save to enter the value for the field. The values in the picklist are predefined by an administrator or developer and stored in the list of values table.

A picklist can be bounded or unbounded. A bounded picklist allows the user to select values from the list only. An unbounded picklist allows users to select values from the list or type values directly into the field.

A static picklist is a selection list that is invoked from a particular text box or list column in an applet. A static picklist in a Siebel application is shown in Figure 73.

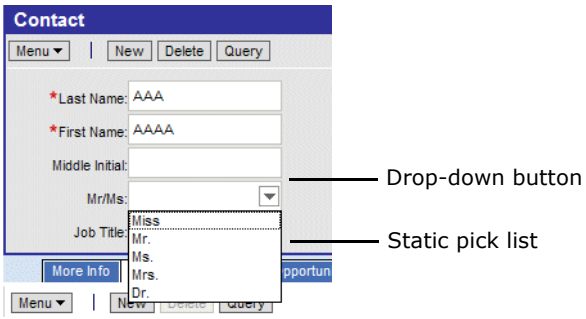


Figure 73. Static Picklist

When the user clicks the drop-down button to the right of the text box, a single-column picklist appears. The user selects a value from the list by clicking the desired value. The selected value replaces the previous value in the text box.

NOTE: You cannot delete the lookup value. You can set the picked field (for example, Lead Quality) back to NULL, unless it is required.

Static picklists differ from dynamic picklists in the following ways:

- They are different in that a static picklist does not draw values dynamically from a pick business component. A static picklist is a static list of available selection values. Configuration of these values is an administration activity that is performed in the List of Values Administration view in a Siebel application.
- They are different in that a static picklist generally does not invoke a dialog box with multiple list columns and buttons. All that appears is a simple one-column pop-up list, without buttons.

  NOTE: It is possible to use a pick applet rather than a simple drop-down list to display a static list of values, but this is not common practice.
- They are different in that a static picklist does not populate multiple controls in the originating applet. It populates a single control in the applet, and the corresponding field in the underlying business component.

For more information about dynamic picklists, see "About Dynamic Picklists" on page 352.

Static picklists are implemented using object types illustrated in Figure 74.

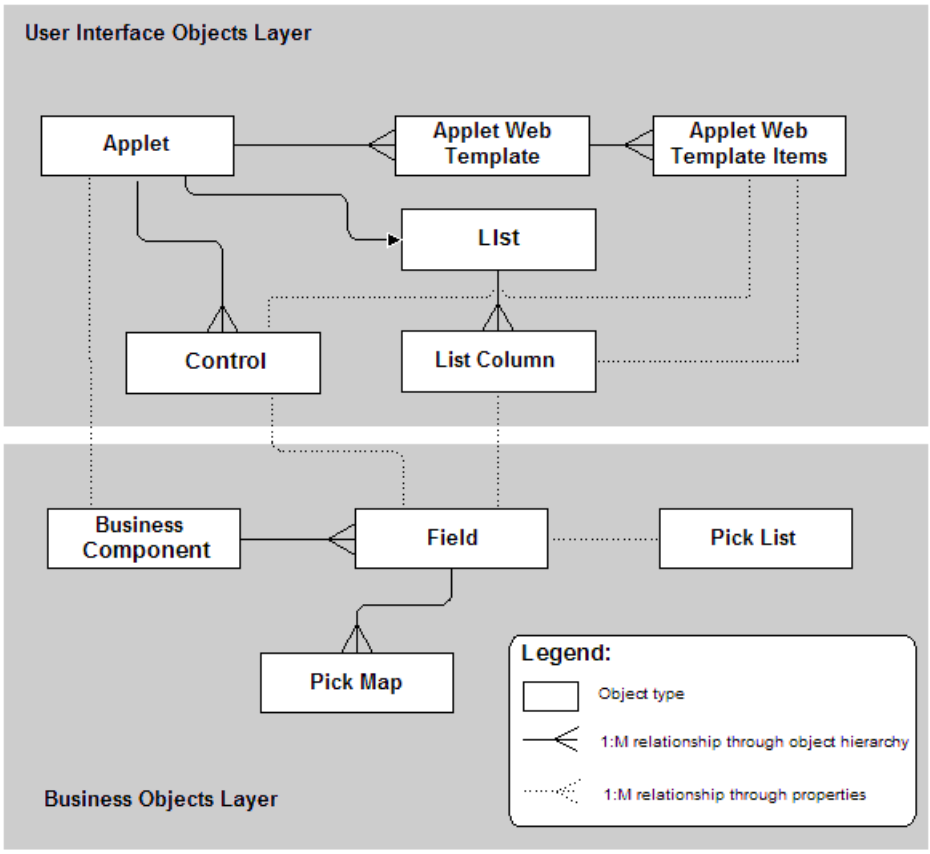


Figure 74. Static Picklist Architecture

Figure 75 shows the object types used in the implementation of a static picklist in greater detail, and identifies their interrelationships.

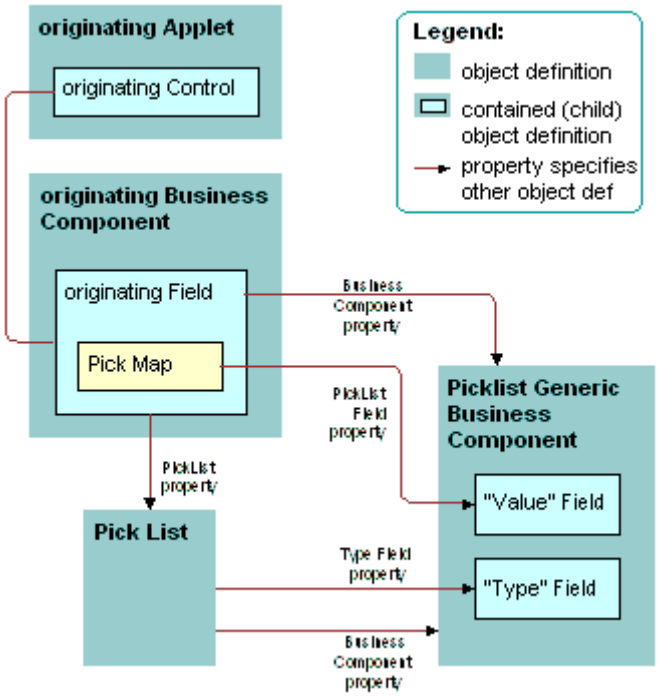


Figure 75. Static Picklist Details

The roles of the object definitions in Figure 75 are summarized in the following list, and discussed in greater detail in the subsequent sections. The static picklist example refers to the Quality picklist illustrated in Figure 73 on page 344.

- **Originating applet.** Contains the control or list column that invokes the picklist. After the selection of a value from the picklist, the originating control displays a revised value. In the example, the originating applet is the Opportunity form applet.
- **Originating business component.** Business component of the originating applet. This business component (in the example, the Opportunity business component) supplies the data presented in the originating applet (Opportunity form applet). The selection process in the picklist results in the update of one field in the current record in this business component.
- **Originating control or originating list column.** Appears in the originating applet. It initiates the picklist when clicked. In the example, this is the Quality control.

- **Originating field.** Field in the originating business component that the originating control represents. Generally, it has one pick map child object definition that defines the mapping of a field from the PickList Generic business component into the originating business component. In the example, the originating field is Quality.

  NOTE: If the originating field is a custom field, make sure that it can accommodate the LOV table values. A field shorter than the LOV table values will cause truncation when it is displayed or stored in the database.

- **PickList Generic business component.** Special-purpose business component for the list-of-value lists that are used in static picklists. It is administered through the List of Values view in the System Administration screen in Siebel applications. To access the List of Values view choose Site Map > Application Administration >List of Values from the menu bar.

- **Pick List object.** The field of the originating control references the Pick List object definition. The Pick List object definition identifies the pick applet's business component, which is always PickList Generic. In the example, the Pick List is called Picklist Quality.

- **Pick Map object.** Child of the originating field. The pick map defines a correspondence between the Value field in the PickList Generic business component and the originating field. This correspondence provides the information required to update the current originating business component record with information from the PickList Generic business component record when a selection is made.

- **Sequence Property.** Defines the sequence for updating fields in the current originating business component record with information from pick business component record when picking this record. If you do not define sequence numbers on pick maps, they will be executed in the order in which they were created.

About the Originating Applet of a Static Picklist

The originating applet (Applet object type) has the following important properties:

- **Business Component.** Identifies the originating business component.

The originating control (Control object type) or list column (List Column object type) has the following important properties:

- **Field.** Identifies the originating field in the originating business component.
- **Pick Applet.** Leave blank for a static picklist.
- **Runtime.** Set to TRUE to indicate that a static picklist is attached, and needs to be activated in response to a user click on the control or list column.

About the Originating Business Component of a Static Picklist

The originating business component is the business component of the originating applet. The data value selected from the pick applet updates the value in the originating field of this business component.

The originating business component has no essential properties for the configuration of a static picklist. However, the field (child) and pick map (grandchild) object definitions are significant.

The originating field is specified in the Field property of the originating Control or List Column object. It has the following important properties:

- **PickList.** Identifies the Pick List object definition.

The originating field has one important child object definition, the Pick Map object. Unlike dynamic picklists, static picklists generally have exactly one Pick Map object definition. The Pick Map object has the following important properties:

- **Field.** Contains the name of the originating (parent) field.
- **Pick List Field.** In this property enter "Value." This setting references the Value field in the PickList Generic business component.

NOTE: You would use multiple pick maps only if you use a multiple column selection list.

About Static Pick List Objects

The Pick List object is referenced by the originating field and identifies the business component and field that populate the pick applet. The Pick List object definition has the following important properties:

- **Business Component.** In this property enter the value PickList Generic. This indicates that the list of values comes from the system tables.
- **Type Field.** In this property enter the value Type. This indicates that Type is the field in the PickList Generic business component to search for types. Each list of values has a type, which uniquely identifies the list and each value in it.
- **Type Value.** In this property enter the relevant type for the list of values. For example, in the Lead Quality picklist in Figure 73 on page 344, the values that appear in the list have a Type field value of LEAD_QUALITY in the List of Values View in Siebel applications.
- **Search Specification.** If a Search Specification value is defined for the Pick List, it overrides the business component's Search Specification. If a Search Specification is not defined, the Search Specification for the business component is used. The default value of the Search Specification is blank.

- **Sort Specification.** If a Sort Specification value appears in the Pick List object definition, this overrides the business component's sorting with that of the Pick List. The default value for the Sort Specification property is blank, which tells the system to use the business component's sorting.

  This feature is useful for nonstandard sorting of values in a static picklist that is based on a list of values in the PickList Generic business component. By default, a list of values is sorted in ascending order on the Order By field within a Type. If the Order By values are blank, the entries for the Type are alphabetically sorted on the Value field, in ascending order. You can alter this behavior for one static picklist by setting a sort specification in its picklist.

- **No Insert.** Static picklists must have their No Insert property set to TRUE to work properly. If this property is set to FALSE the application generates the following error message:

  "Unable to create picklist popup applet."

Creating a Static Picklist Using the Pick List Wizard

You can create a static picklist by using the Pick List Wizard.

To create a static pick list using the Pick List wizard

1. From the Tools main menu, choose File > New Object.

   The New Object Wizards dialog box appears.

   NOTE: You can also invoke the Pick List wizard by selecting the field for which you would like to create a picklist, right-clicking, and choosing Add Pick List.

2. Select the Pick List icon, and click OK.
3. In the Pick List dialog box, enter the following information, and then click Next.
   - Project
   - Business Component (originating business component; the parent business component of the field that will display the picklist)
   - Field
4. In the Pick List Type dialog box, select Static, and then click Next.

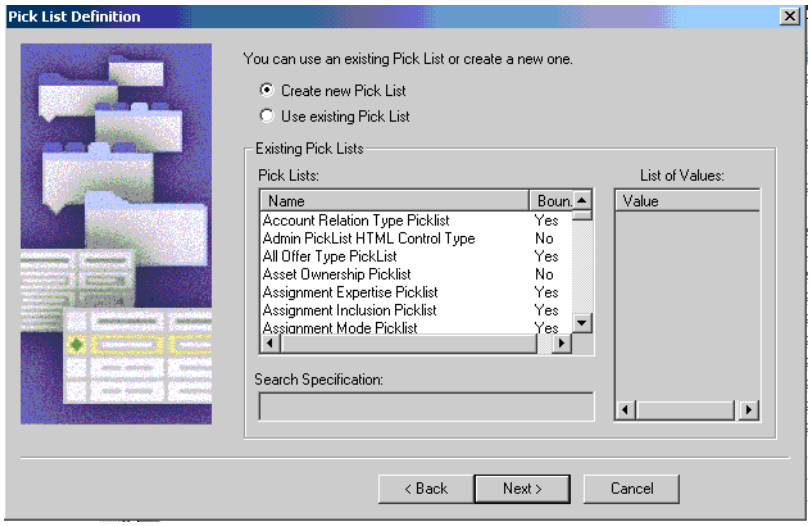
   NOTE: Static picklists draw their values from a predefined list of values (LOV). Dynamic picklists draw their values from a business component. For more information about dynamic picklists, see "Creating Dynamic Picklist Objects" on page 365.

5. In the Pick List Definition dialog box, do one of the following:
   - If you want to create a new picklist, select the Create New Pick List option button and then click Next.

     The Pick List Definition dialog box appears.

- If you want to use an existing picklist, select the picklist and associated list of values you want to use, and then click Next.

  The Finish dialog box appears. Go to Step 8.



6 If you are creating a New List of Values, do the following:

   a Enter a unique name for the picklist.

   b Select the Create New List of Values option button, and then click Next.

   c In the List of Values dialog box, enter a name for the List of Values and the Values.

      For more information about List of Values, see *Applications Administration Guide*.

   d Click Next.

7 If you are using an existing List of Values, do the following:

   a Enter a unique name for the picklist.

   b Select the Use predetermined List of Values radio button.

   c Select the List of Values Type, and then click Next.

   d In the third Pick List Definition dialog box, enter a search specification, a comment, and select whether you want the picklist to be bounded.

   e Click Next.

8 In the Finish dialog box, review the specifications for the picklist, and then click Finish.

About the Picklist Generic Business Component

The PickList Generic business component is a specialized business component reserved for lists of values for static picklists. The data in the Picklist Generic business component looks something like Table 50.

Table 50. Example of Data in Picklist Generic Business Component

Type Field Contents	Value Field Contents
LEAD_QUALITY	Excellent
LEAD_QUALITY	Very Good
LEAD_QUALITY	High
LEAD_QUALITY	Fair
LEAD_QUALITY	Poor
PERSON_TITLE	Mr.
PERSON_TITLE	Ms.
PERSON_TITLE	Dr.
ACCOUNT_TYPE	Commercial
ACCOUNT_TYPE	Competitor
ACCOUNT_TYPE	Customer

Two of the fields in the Picklist Generic business component together define and group the lists of values, as follows:

- **Type.** Each list of values has a type. The type groups together all records that are in one list of values. For example, a type of LEAD_QUALITY identifies a record as a member of the Lead Quality list of values, and the type ACCOUNT_TYPE refers to the Account Type list of values.
- **Value.** The Value is the portion of the record that actually appears in the static picklist. For example, Lead Quality values are Excellent, Very Good, High, Fair, and Poor.

About Dynamic Picklists

Like static picklists, dynamic picklists allow the user to populate fields by selecting values from a list. However, rather than drawing the values from the list of values table, a dynamic picklist draws its values from another user-maintained business component. Fields that use dynamic picklists are typically joined fields displaying data from a table other than the business component's base table. The dynamic picklist allows users to update the joined field.

Dynamic Pick Lists are exposed in the user interface using *Pick applets.* Pick applets allow users to select a value from a list, and have the selection entered into controls or list column cells (Figure 76).

NOTE: In end-user documentation, pick applets are referred to dialog boxes.

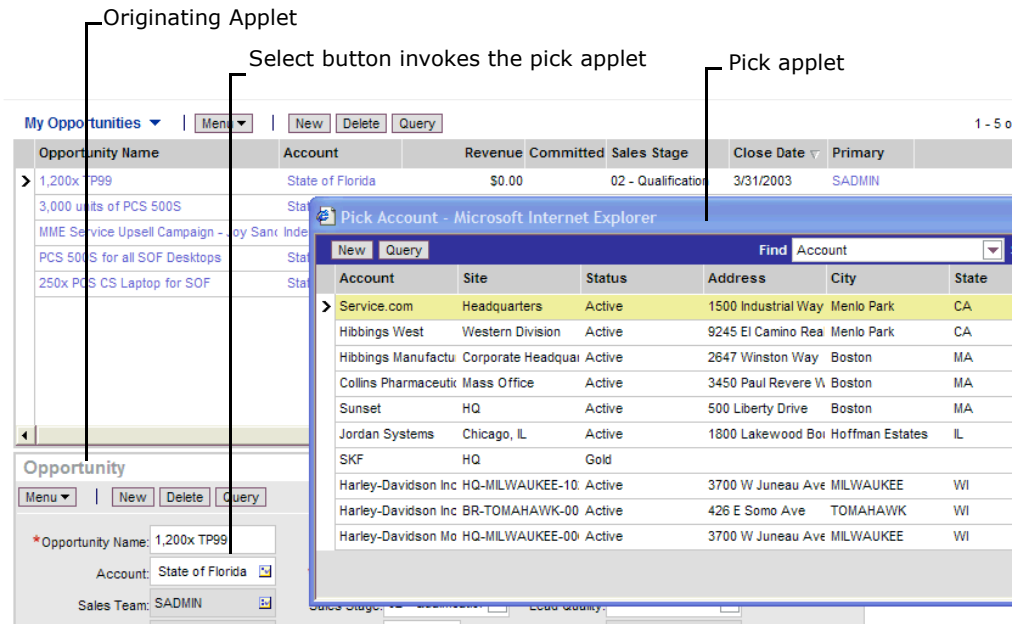


Figure 76. Pick Applet

The data in the pick applet typically comes from a different business component than the data in the originating applet. There can be exceptions, for example, picking a parent Account for an Account or a parent Position for a Position record. When the user selects a record in the pick applet, the values in certain list columns in the selected record are copied to corresponding list columns in the originating applet. This is illustrated in Figure 77.

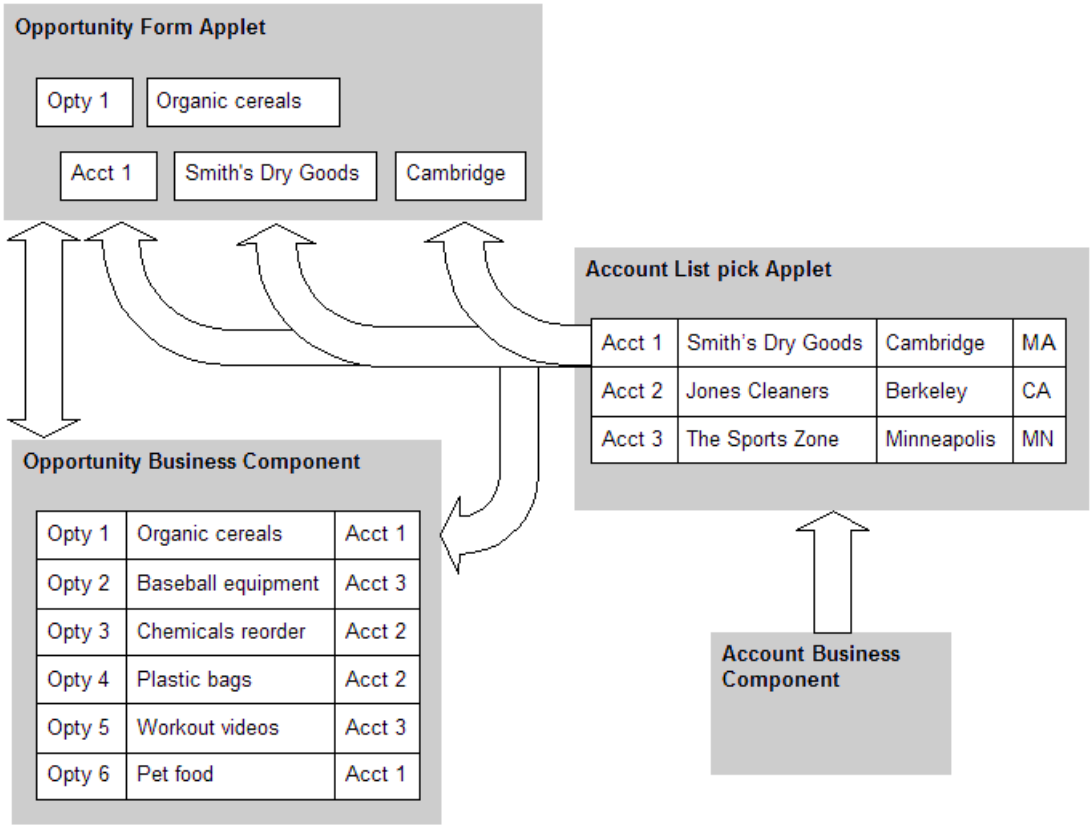


Figure 77. Data Flow in a Pick Applet

The following steps take place, from the user's perspective:

1. In the Opportunity Form applet, the user enters information for the Organic Cereals opportunity.
2. In the Opportunity Form applet, the user clicks Select.

   The Account pick applet appears and displays rows from the Account business component.
3. The user selects Account 1, Smith's Dry Goods, in the pick applet, and then clicks OK.
4. Account data for Smith's Dry Goods populates controls in the Opportunity Form applet.

Dynamic Pick Lists maintain the foreign keys that facilitate join relationships. In the opportunity and account example, there is a foreign key in the Opportunity business component identifying the account for each opportunity. When the user selects an account in the pick applet, it populates this foreign key field. This selection associates the account with this opportunity for future use by the join that uses the foreign key.

Dynamic Pick Lists are implemented using object types illustrated in Figure 78.

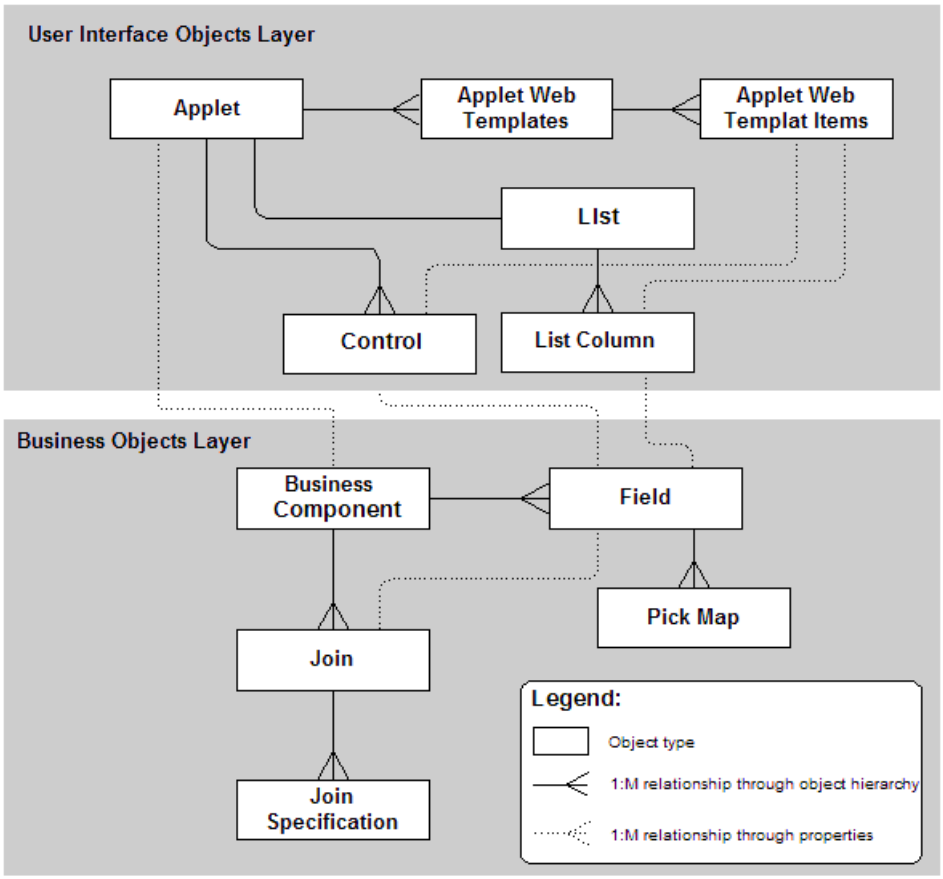


Figure 78. Pick Applet Architecture

Figure 78 shows the object definitions used in the implementation of a pick applet in greater detail, and identifies the interrelationships.

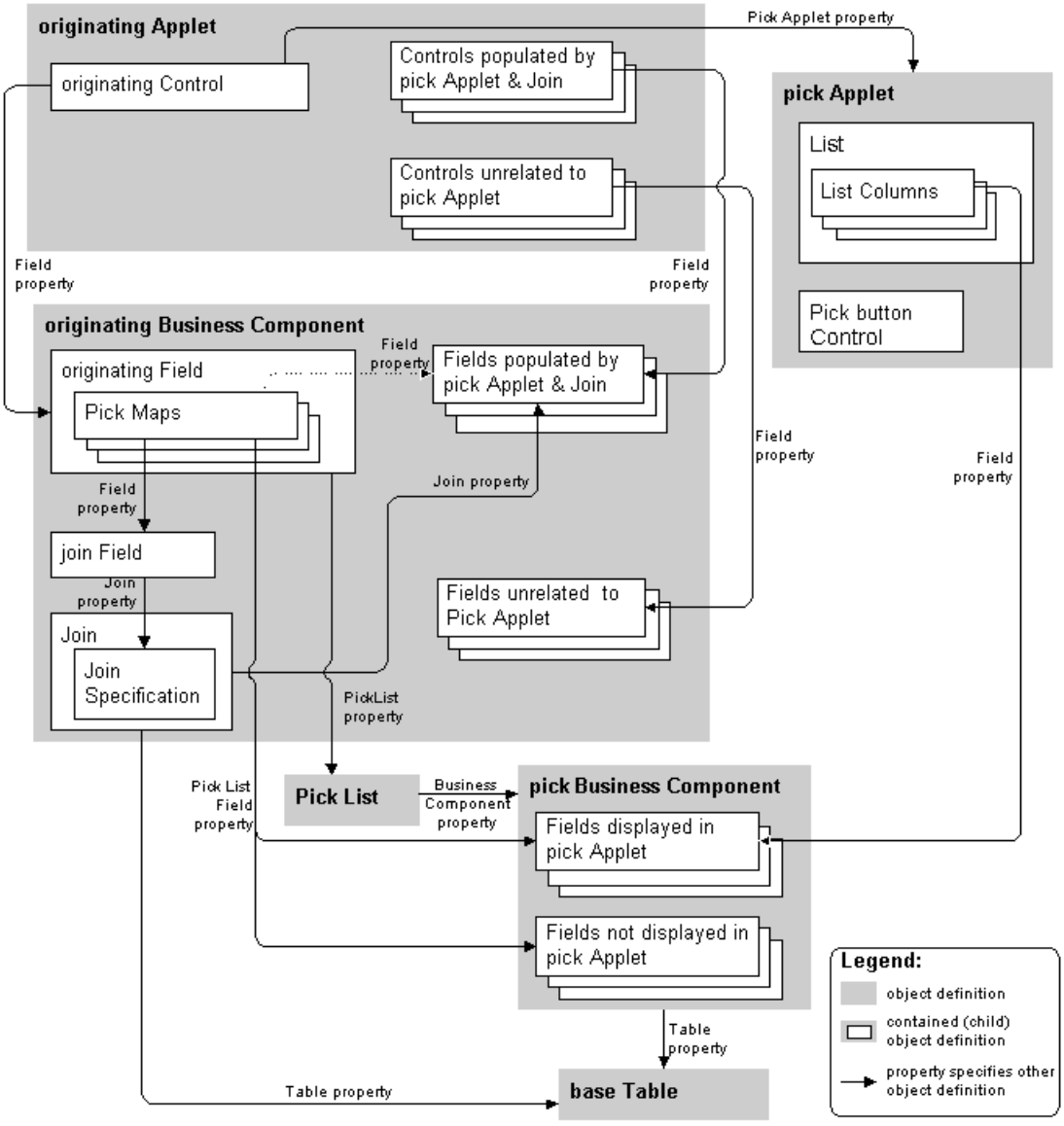


Figure 79. Pick List

The roles of the object definitions in Figure 79 on page 355 are summarized in the following list and discussed in greater detail in the subsequent subsections. The pick applet example referenced is the Account pick applet illustrated in Figure 76 on page 352.

- **Originating applet.** Contains the control or list column that invokes the pick applet. After the pick applet is invoked and a value is selected, specific controls in the originating applet display revised values. In the example the originating applet is the Opportunity Form Applet.
- **Pick applet.** Dialog box that is invoked for the selection of a value. The dialog box is a list applet containing scrolling list table rows. Each row corresponds to a business component record. In the example, the pick applet is called Account pick applet.
- **Originating business component.** Business component of the originating applet. This business component (in the example, the Opportunity business component) supplies the data presented in the originating applet (Opportunity form applet). The selection process in the pick applet results in the update of the current record in this business component.
- **Pick business component.** Business component of the pick applet. Data from fields in this business component is displayed in the list columns of the pick applet. In the example the pick business component is Account.
- **Originating control or originating list column.** Appears in the originating applet. When you click the originating control or list column, it invokes the pick applet. In the example, the originating control is the Account control.
- **Originating field.** Field in the originating business component that the originating control represents. It has pick map child object definitions that define the mapping of fields from the pick business component into the originating business component. In the example the originating field is the Account field.
- **Pick List.** Referenced by the field of the originating control, and identifies the pick applet's business component. In the example, the picklist object is called PickList Opportunity Account.
- **Pick maps.** Children of the originating field. Each pick map object definition defines a correspondence between a field in the pick business component and one in the originating business component. These correspondences provide the information required to update the current originating business component record with information from the pick business component record as soon as a record is picked.

  When a user selects a value from an unbounded picklist, it is copied to the field with which the picklist is associated using the corresponding pick map that references the same field. Fields associated with other child pick maps are only populated when the picklist is bounded.

  NOTE: Typing a new value into an unbounded picklist does not automatically add it to the list of values that can be picked.

  Fields in pick map objects are not updated when a user picks a value from an unbounded picklist. Any applet based on CSSBuscomp or CSSBCBase with an unbound picklist will not map all the values in the pick map. For all the values in a pick map to be mapped, the picklist must be bounded.
- **Join and join specification.** Child object definition of the originating business component. The join specification is a child of the join and is referenced by the join field. One of the pick maps updates the join field. A change in the value of the join field results in the update of all fields whose values are derived from the join. This update is not as immediate as the update performed by the pick maps. In the absence of the other pick maps, the data would not be updated until the user left the view and returned to it. In the example the join is called S_ORG_EXT and the join specification is Account Id.

About the Originating Applet of a Dynamic Picklist

The originating applet contains the control or list column that invokes the pick applet. It may also contain other controls or list columns that are populated by the user's selection from the list applet. The originating applet itself requires no special configuration.

Figure 80 is a detail of the originating applet in Figure 78 on page 354.

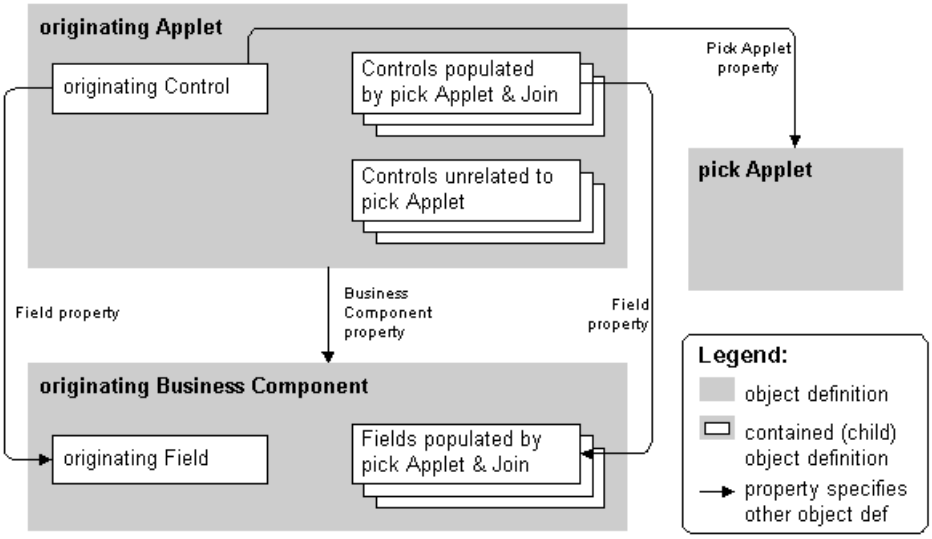


Figure 80. Originating Applet Details

As indicated in Figure 80, the important property setting for the originating applet is as follows:

- **Business Component.** Creates the association between the originating applet and the originating business component.

The important child object definitions of the originating applet are the following:

- **Originating control.** Invokes the pick applet, as the result of the user's clicking the drop-down arrow. The originating control has the name of the pick applet in its Pick Applet property. The field specified in the Field property of the originating control is the originating field, and has pick map child object definitions, as discussed in "About the Originating Business Component of a Dynamic Picklist" on page 362.

  The control or list column must have its Runtime property set to TRUE.

- **Controls populated by the pick applet.** Each control for which some field in the originating business component is populated by a pick map object definition will be updated when the user makes a selection from the pick applet.

- **Controls unrelated to the pick applet.** Other controls in the applet.

About Pick Applets

Pick applets are invoked by clicking the Select button that appears next to certain fields. Pick applets contain a scrolling list table of available selections in one list column, with the information from related fields in adjacent list columns. The user selects a row in the list table and clicks the OK button to accept the selection. Each row corresponds to a business component record in the pick business component. The pick applet is dismissed, and the user's selection populates the text box or list column cell in the originating applet (the applet from which the pick applet was invoked). The user's selection can also populate other controls or list column cells in the originating applet.

For example, when a user clicks the Select button in the Account field in the Opportunity Form applet, the Pick Account dialog box (pick applet) appears for the selection of an account. When an account has been selected, and the Pick Account dialog box has been dismissed, the Account text box contains the selected account, and the Site text box in the originating applet contains the site that corresponds to the selected account.

Figure 81 shows a detailed definition of the pick applet from Figure 78 on page 354.

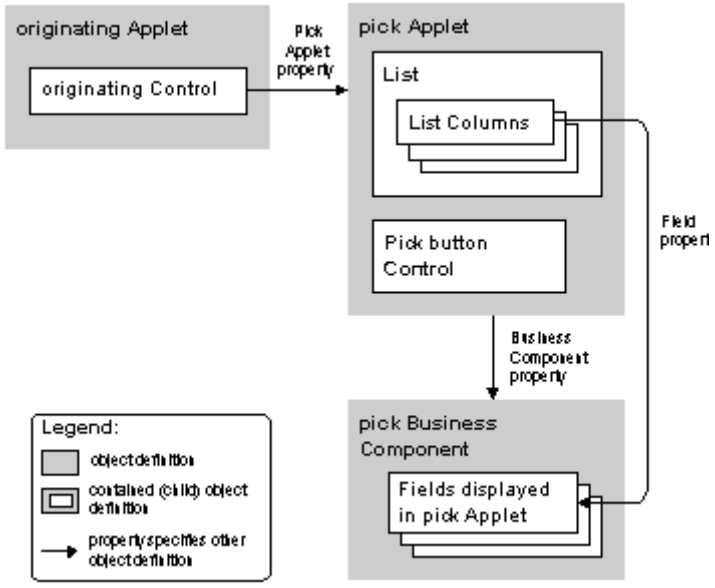


Figure 81. Pick Applet Details

The pick applet (Applet object type) has the following important property settings:

- **Business Component.** Pick business component.
- **Class.** CSSFrameList, indicating that this is a list applet.
- **Type.** A value of Pick List is entered, to indicate that this is a pick applet. This setting configures the behavior of the dialog box and button controls.
- **Title.** Name of the pick applet that appears in the title bar.

The pick applet has the following important child object definitions:

- **List.** List columns are attached to the list.
- **List columns (grandchild object definitions).** Each displays the contents of one field in the business component.
- **Pick Record control.** Invokes the PickRecord method when clicked. The PickRecord method locates the pick map child object definitions of the originating field and, from these, determines which fields to update in the originating business component. These fields are updated based on the record selected from the pick business component by the user.
- **Web Templates.** Define the layout, such as position of list columns and controls, for each of the defined modes.
- **Web Template Items.** Map list columns and controls to placeholders in the Web template. Web template items exist for each list column and control defined for the applet.

Creating Pick Applets

Often when implementing dynamic picklist, you can use an existing pick applet to expose the data in the user interface. However, if an existing pick applet does not meet your requirements, you can use the Pick Applet Wizard to create a new one.

Always name Pick applets <BusComp> Pick Applet. Always name association applets <BusComp> Assoc Applet.

To configure applets using the Pick Applet wizard

1. From Siebel Tools main menu, choose File > New Object.

   The New Object Wizards dialog box appears.

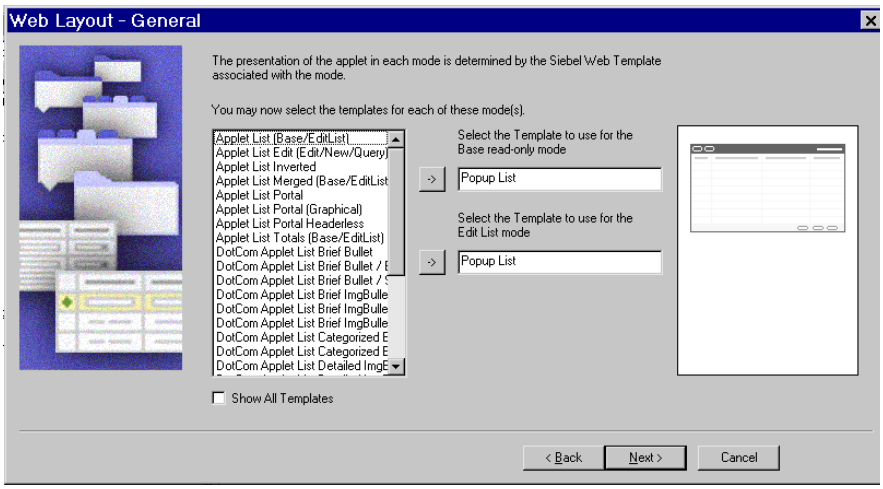
2. Click the Applets tab.
3. Click the Pick Applet icon, and then click OK.

   The General dialog box appears.

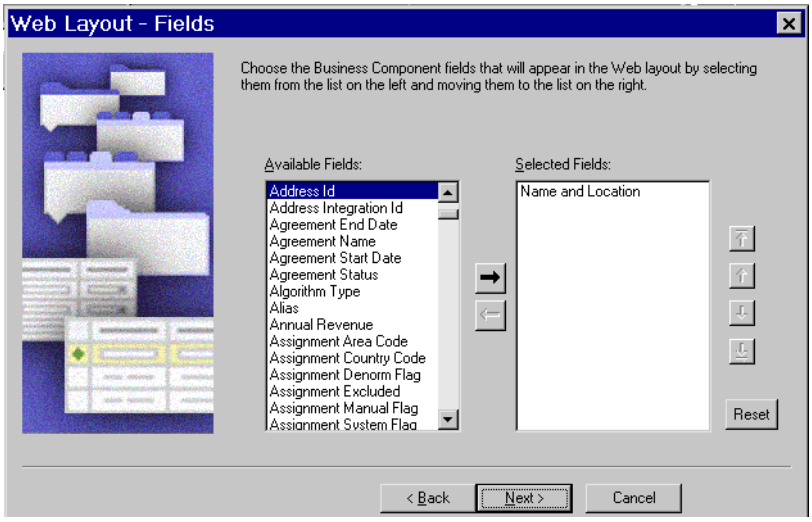
4. In the General dialog box, enter information for the following and then click Next.
   - Project
   - Pick business component
   - Name for the Picklist Applet
   - Display Name

   The Web Layout General dialog box appears.

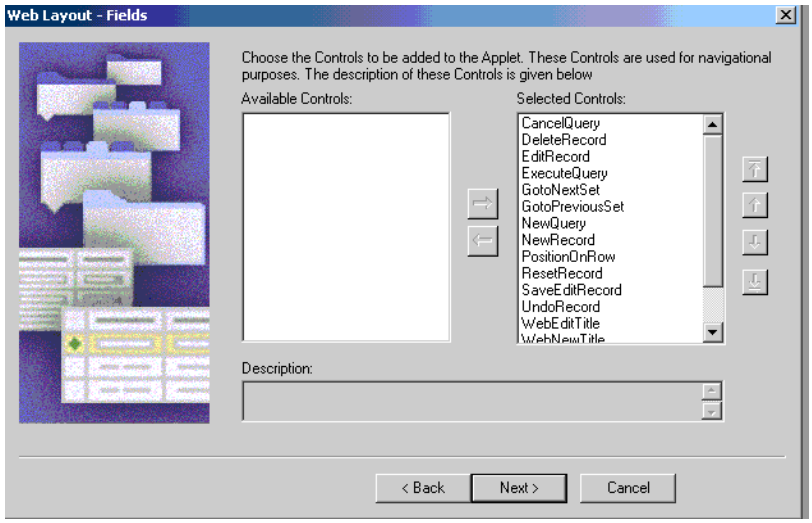
5 In the Web Layout General dialog box, select the templates to use for the Base and Edit List modes, and then click Next.



6 In the Web Layout - Fields dialog box, select the fields you want to appear in the pick applet, and then click Next.



7 In the Second Web Layout - Fields dialog box, select the controls you want to appear in the pick applet and then click Next.



NOTE: By default all the controls are present in the Selected Controls box. If you wish to deselect any of these controls, highlight them and click the left arrow to move these controls into the Available Controls box. The controls that appear by default are based on the Model Pick Applet in the Siebel repository.

8 In the Finish dialog box, review the information, and then click Finish.

The Pick Applet Wizard creates the pick applet object, and then opens the Web Layout editor that you can use to map list columns and controls to the placeholders in the Web Template.

For more information, see "Editing Applet Layout" on page 259.

About the Originating Business Component of a Dynamic Picklist

The originating business component is the business component of the originating applet, as specified in the Business Component property of the Applet object. This business component supplies the data presented in the originating applet. The selection process in the pick applet results in the update of the current record in this business component. Figure 82 on page 362 shows the detailed definition of the originating business component from Figure 78 on page 354.

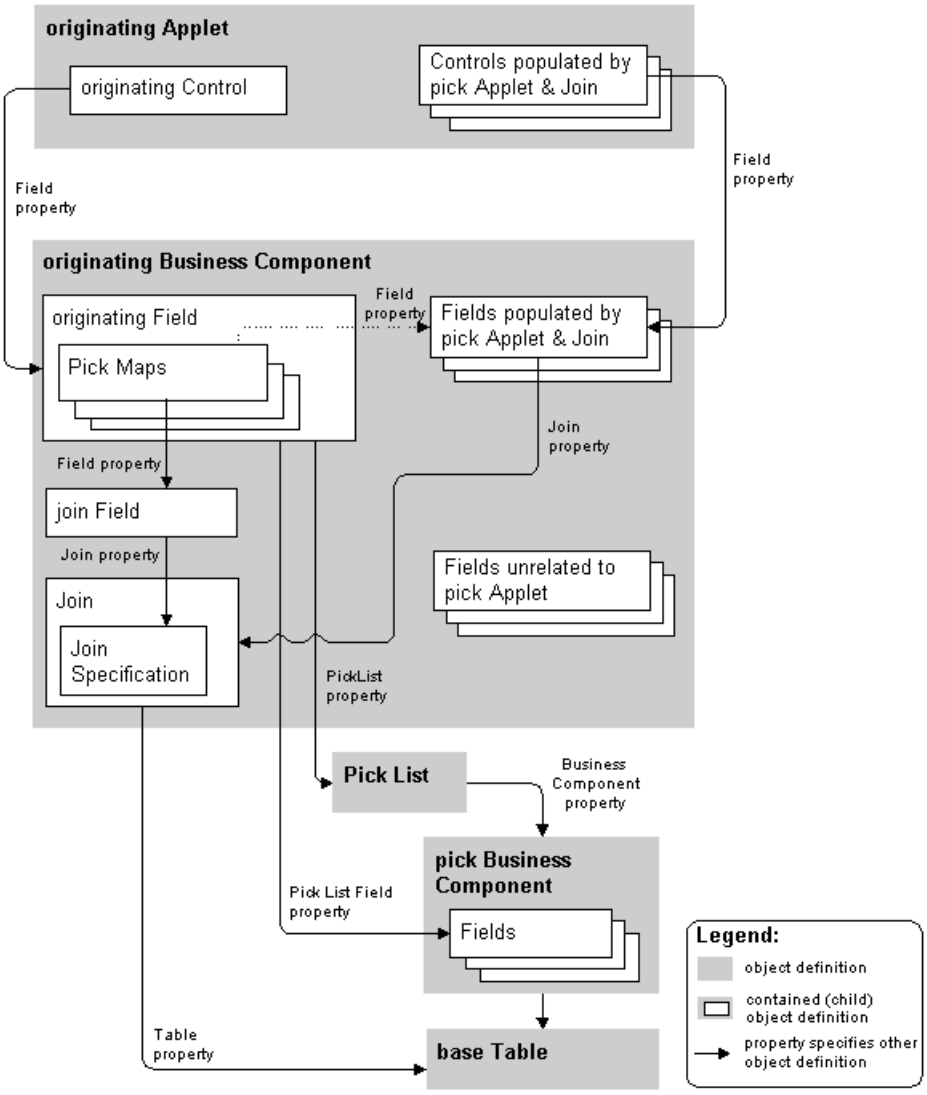


Figure 82. Originating Business Component Details

The originating business component has no important property settings that are related to its role in the pick process.

The originating business component has the following important child object definitions:

- **Originating field.** The originating control displays the data from this field. The originating field has no special role other than being the parent of the pick map object definitions. The Pick List property of the Field specifies the Pick List object. In the Siebel application architecture, pick maps are children of an originating field, rather than the originating business component, in order to support pick applets on more than one field in the business component.

  NOTE: The originating field should be a field based on a database column. Pick applets and picklists cannot be associated with read-only fields, including calculated fields.

- **Pick maps.** Children of the originating field. Each pick map defines a correspondence between a field in the pick business component and one in the originating business component. These correspondences provide the information required to immediately update the current originating business component record with information from the pick business component record when a record is picked. Additionally, one of the pick maps updates the join field, and eventually this causes the join to update the fields in the business component that are dependent on the join.

  NOTE: Test your pick map definition after creating it. If the originating field stays the same after choosing a value from the pick applet, you should check the pick map definition for that field.

  Each Pick Map object definition has two important properties:

  - **Field.** Identifies a field in the (grandparent) originating business component that is to be populated by data from a field in the pick business component, when the PickRecord method is invoked.
  - **Pick List Field.** Identifies a field in the pick business component that is the source of data for the field in the Field property of the Pick Map object.

    Fields in Pick Map objects are updated when the user picks a value from an unbounded picklist. However, fields in Pick Map objects are not updated by the picklist when the user types in a new value (the field the user typed something into is, however, obviously updated with the user's entry).

    Typing a new value into an unbounded picklist does *not* automatically add it to the list of values that can be picked.

    Do not define more than one multi-value field in the originating business component that maps to the same destination field in the pick applet's underlying business component (Pick List Field Property). Doing so causes the drop-down arrow for the picklist not to show; as a result, users will not be able to use the picklist.

- **Join field.** Serves as a foreign key in the join used by the pick applet. Typically, the join field contains Id in its name, such as Account Id or Key Contact Id. It is identified in the Source Field property of the join specification. The join field is one of the fields identified in a pick map object definition. When the user selects a record from the pick applet, the join field is updated (because of the pick map in which it is identified), and this results in the update of all fields that are based on the join.

  NOTE: Fields in the originating business component, and the controls or list columns that represent them, initially are updated by the action of the pick maps. The join and join specification do not update the contents of the applet until the user leaves the view and returns to it.

- **Join and join specification.** The join and join specification object definitions set up the join between the base tables of the originating and pick business components. This join populates those fields in the originating business component that have this join's name in their Join property.
- **Fields populated by the pick applet and join.** Fields that have the join's name in their Join property are updated when the join field's value changes. Fields that are identified in the Field property of Pick Map object definitions are updated when a selection is made from the pick applet. There is some overlap in the roles of the pick maps and join, in that both generally update the same fields, but the action of the pick maps is immediate and that of the join is somewhat delayed.

  That is, even though pick maps can update the display value of joined fields (for example, Account Name) when the user picks a record, pick maps do not physically copy a value to the joined fields—only to the foreign key field (for example, Account Id).

About Dynamic Picklist Objects

The field of the originating control references the Pick List object definition. The Pick List object definition identifies the pick business component. In this way, the identity of the pick business component is made known to the pick applet.

Figure 83 shows the detailed definition of the Pick List object definition from Figure 78 on page 354.

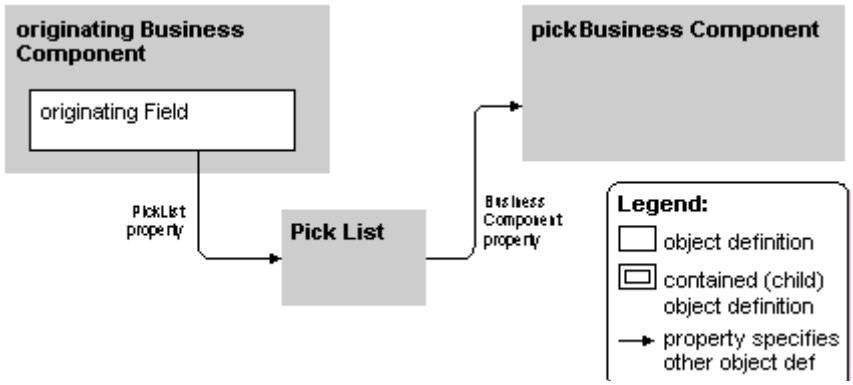


Figure 83. Pick List Details

The Pick List object definition has the following important property, when used in pick applet configuration: Business Component. This property identifies the pick business component.

NOTE: When configuring a pick applet invoked from a multi-value group applet, define the picklist on the originating field in the originating business component, not on fields in the multi-value group business component. For more information on multi-value group applets, see "Configuring Multi-Value Group and Association Applets" on page 401.

Creating Dynamic Picklist Objects

The Pick List Wizard walks you through the process of creating a dynamic pick list and related objects, which include:

- **Pick List.** Object that defines the properties of the picklist, including the originating business component and the pick business component.

  NOTE: The originating business component is the one on which you are creating the dynamic picklist. In the current example, it is the Action business component. The pick business component is the one from which you are picking values to display to the user. In the current example, it is the Opportunity business component.

- **Pick Maps.** Child object of a business component field that map the source field in the pick business component with the target field in the originating business component.

- **Pick Applet.** Pop-up applet that allows you to display the list of records from which the user can select.

Always name a new PickList object ABC PickList <entity>. Note that if the entity name itself has a prefix, it does not need to be repeated. For example, a PickList based on the MS Subsegment business component would be ABC PickList Subsegment instead of MS PickList MS Subsegment.

NOTE: The values of the Visibility Type and Visibility Auto All properties of the picklist object override the pop-up visibility properties on the business component. For detailed information, see *Security Guide for Siebel eBusiness Applications*.

To create a dynamic picklist using the Pick List wizard

1. From the Tools main menu, choose File > New Object.

   The New Object Wizards dialog box appears.

   NOTE: You can also invoke the Pick List wizard by selecting the field for which you would like to create a picklist, right-clicking, and choosing Add Pick List.

2. Select the Pick List icon, and then click OK.

   The Pick List Wizard appears.

3. In the Pick List dialog box, enter the following information and then click Next.

   - Project
   - Business Component (originating business component; the parent business component of the field that will display the picklist)
   - Field

4. In the Pick List Type dialog box, select Dynamic.

   NOTE: Static picklists draw their values from a predefined list of values (LOV). Dynamic picklists draw their values from a business component. For more information about static picklists, see "Creating a Static Picklist Using the Pick List Wizard" on page 349.

5. In the Pick List Definition dialog box, choose whether you want to create a new picklist or use an existing one:

- If you want to create a new picklist, select the Create a New Pick List option button, and then click Next.

  The Pick List Definition dialog box appears. Go to Step 6.

- If you want to use an existing picklist, select the Use Existing Pick List option button, select the picklist from the Pick List box and then click Next.

  The Pick Map dialog box appears. Skip to Step 7.

6 In the Pick List Definition dialog box, enter the information for the picklist, and then click Next.

- Business Component (pick business component)
- Sort field in the picklist
- Name
- Search Specification (not required)
- Comment (not required)

7 In the Pick Map dialog box, select the source field in the originating business component and the target field in the pick business component, and then click Add.

The selected fields are displayed in the Current Map window.

8 Click Next, verify the information in the Finish Dialog box, and then click Finish.

Constraining Dynamic Picklists

You can dynamically filter a pick applet to display only records that have field values matching corresponding fields in the originating business component's records. This is called constraining a picklist. For example, a Contact's pick applet invoked from an applet that displays quotes could be configured to display only contacts for the current quote's account.

Pick applet constraints are defined using the Constrain property in the Pick Map object type. For example, if you want to configure a Country picklist to display only states that are part of that country, you need a way to indicate the relationship between each state and its country. You could use the existing Description field in the Picklist Generic BusComp to do this or alternatively, you could extend the table and use a new column. Next, you would need to fill the Description field with valid Country values. Use one of the following methods to accomplish this.

- From the client application, choose Site Map > Application Administration > List Of Values View and populate the Description field with valid Country values.
- From Siebel Tools, choose Account BusComp > State Field > PickMap. Insert a new record in the PickMap list and set the following properties:
  - Field = Country
  - Constrain = True
  - Pick List Field = Description

After a user selects a value from the Country picklist, the State picklist appears. The values in the State picklist are constrained by the value selected from the Country picklist. The value chosen from the Country picklist is used to filter the values that appear in the State picklist. Only the values where the Description field contains the selected value from the Country picklist will appear in the State picklist.

Pick maps can be of two types: copy pick map or constraint pick maps. Copy pick maps perform the role described in "About the Originating Business Component of a Dynamic Picklist" on page 362: a copy pick map updates the current originating business component record with information from the pick business component. A constraint pick map also configures a mapping between the originating and pick business components, but its purpose is different. It is used to filter the list of records displayed in the pick applet to present only those that have matching values in corresponding fields in the originating and the pick business component.

A pick map is configured as a constraint pick map by setting its Constrain property to TRUE. If FALSE (which is the default), the pick map is a copy picklist.

The pick applet displays only contacts with the same Account, Account Id, and Account Location as the quote. To accomplish this, define a constraint pick map as a child object of the Contact Last Name field (in addition to the various copy pick map object definitions provided in order to implement pick behavior). The presence of this constraint pick map indicates to the system that it is to filter the displayed records in the pick applet.

NOTE: If the constrained field refers to a joined table in the pick business component, the foreign key field must also be constrained. Otherwise, a "This operation is not available for read-only field" error will occur if a new record is created in the pick applet.

The pick business component is the business component of the pick applet. Data from fields in this business component is displayed in the list columns of the pick applet.

Figure 84 shows a detailed definition of the pick business component from Figure 78 on page 354.

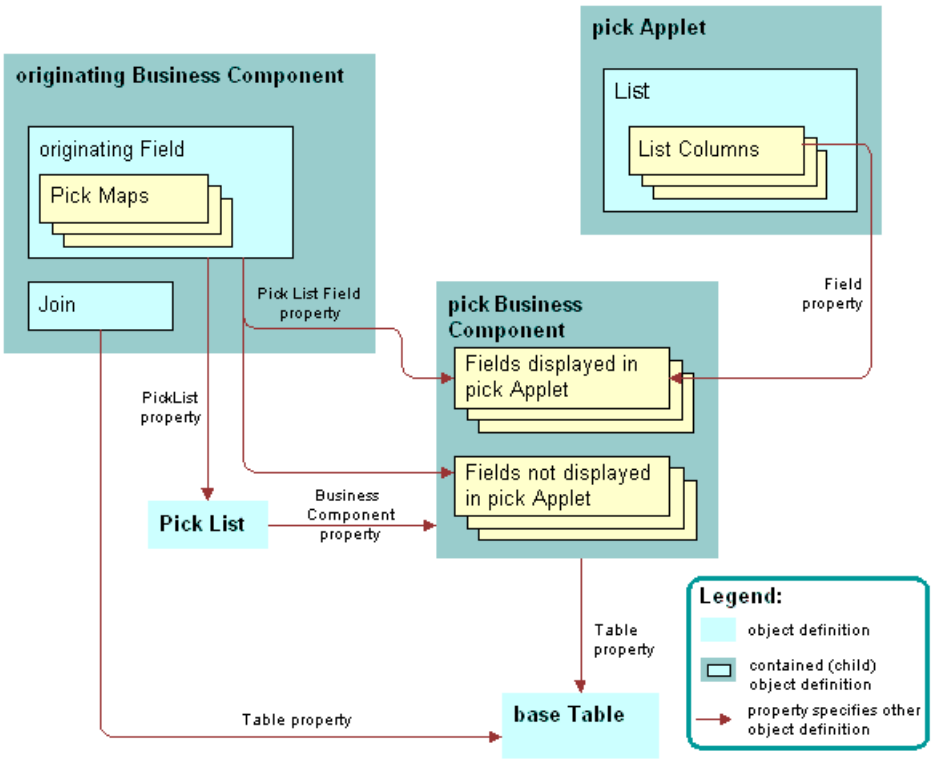


Figure 84. Pick Business Component Details

The pick business component has no important property settings with respect to its role in the pick process.

The pick business component has the following important child object definitions:

- **Fields displayed in the pick applet.** Populate the list columns in the pick applet. They are referenced in the Field property of corresponding list columns in the pick applet. Some of the same fields may be identified in the Pick List Field property of Pick Map object definitions and, hence, have a role in updating corresponding fields in the originating business component.
- **Fields not displayed in the pick applet.** Although not displayed in list columns in the pick applet, some of these fields may be identified in the Pick List Field property of Pick Map object definitions and therefore have a role in updating corresponding fields in the originating business component.

About Hierarchical Picklists

A hierarchical picklist displays values that are constrained by values selected in another picklist. For example, in the Service Request Detail Applet, the Area and Subarea fields are both picklists that draw their values from the List of values table (S_LST_OF_VAL). The items available in the Subarea picklist depend on what the user has selected in the Area picklist.

The hierarchical relationship between the values is established in the list of values table. All the values for picklists within the hierarchy are defined using the same LOV Type. For example, for Area and Subarea, the values are defined using the SR_AREA LOV Type.

The Parent Independent Code column is used to specify a parent value. For example, consider the following example LOV shown in Table 51.

Table 51. Sample LOVs for Hierarchical Picklist

Type	DIsplay Value	Language Independent Code	Parent LIC
SAMPLE_LOV	1	1	
SAMPLE_LOV	A	A	1
SAMPLE_LOV	B	B	1
SAMPLE_LOV	2	2	
SAMPLE_LOV	C	C	2
SAMPLE_LOV	D	D	2

Assume two picklists are configured to display the values shown in Table 51 in a hierarchical relationship. One picklist is the parent picklist, and the other picklist is the child. The parent picklist displays the values {1, 2}. If the user selects 1, the values displayed in the child picklist are {A, B}; If the user selects 2, the values displayed in the child picklist are {C, D}.

Implementing a hierarchical list of values also involves configuration work. You must configure two picklists to support this hierarchical relationship. The parent picklist must be based on the PickList Hierarchical business component, and the child picklist must be based on the PickList Hierarchical Sub-Area business component.

Configuring a Hierarchical Picklist

To configure a hierarchical picklist

1. Configure a parent and a child picklist.
   a. Set the Business Component property of the parent picklist to Picklist Hierarchical.
   b. Set the Business Component property of the child picklist to PickList Hierarchical Sub-Area.

2 Go to the business component that contains the fields that you want to associate the hierarchical picklists with.

   a Set the Picklist property of the parent field to the parent picklist.

   b Set the Immediate Post Changes property of the parent field to TRUE.

   c Set the PickList property of the child field to the child picklist.

   d For the child field, create the following Pick Map objects.

Field	PickList Field	Constrain
[name of parent field]	Parent	TRUE
[name of child field name]	Value	

3 Compile changes to a repository file.

4 Add LOV values using the Parent LIC column to designate the parent value.

   See table above for a simple example and see the Constrained Lists of Values section in *Applications Administration Guide* for a detailed discussion.

5 Test.

18 Creating and Administering Lists of Values

Topics in This Chapter

About Lists of Values

Lists of values are sets of values that populate static picklists. When a user selects a static picklist, a list of values is displayed. The user can select a value from the list to populate the field.

The values in a LOV are stored as records in the database (S_LST_OF_VAL). They are grouped together using the Type field. For example, the values of LOV that appear in the Status field of the Account Entry Applet all have a Type value of ACCOUNT_STATUS. Table 52 lists several values, all belonging to the same LOV Type.

Table 52. Sample Values from the ACCOUNT_STATUS LOV

Type	Display Value
ACCOUNT_STATUS	Candidate
ACCOUNT_STATUS	Qualified
ACCOUNT_STATUS	Active
ACCOUNT_STATUS	Inactive

Picklist objects in the Siebel repository include a Type property that identifies the LOV Type associated with the picklist. This object definition is compiled to the SRF. At run time, the Siebel application reads this information to identify which group of LOVs to display for a given picklist in the user interface.

Creating New LOV Types and LOV Values Using Siebel Tools

Siebel applications come with many LOVs that support the static picklists used throughout the Siebel user interface. However, your organization may require new values added to existing LOV Types or you may require entirely new LOV Types.

Typically, administrators use the List of Values administration views in the Siebel run-time client to add, modify, or inactivate LOV values for existing LOV types. For example, an administrator might add an additional value to the ACCOUNT_STATUS LOV Type. For information about working with existing lists of values, see *Applications Administration Guide*.

However, if you need to add a new set of values, that is, define a new LOV Type, you typically perform this task in Siebel Tools. This is because to implement a new LOV Type, you must also associate LOV records to a picklist object defined in the repository and compile that information to an SRF file for the run-time client to read.

To create a new list of values type using Siebel Tools

1 Choose Screens > System Administration > Lists of Values.

Lists of values are displayed in the Object List Editor.

2 Create a record for a new LOV Type using the information in the following table.

Field	Description
Type	Enter LOV_TYPE. Records with a value in the Type field of LOV_TYPE are used to group LOV Values of the type.
Display Value	Enter the name of the LOV Type. For example, ACCOUNT_STATUS. This value is used as the value in the Type field for all child records of this type.
Language Independent Code	Typically the same value as the American English version of the display value. See "About the Language Independent Code" on page 378.
Translate	Information-only field used by Siebel Engineering. There is no client or server functionality associated with this field.

NOTE: To see all LOV Types, query for records with the Type field equal to LOV_TYPE.

3 Enter new records for the LOV values using the information in the following table.

For a complete description of LOV fields, see *Applications Administration Guide*.

Field	Description
Type	The name of the list of values type, for example ACCOUNT_STATUS. This value is used to group all other values for this type. The value defined in this field must match the value defined in the Type Value property of the picklist that is configured to display these values.
Display Value	Value displayed in the picklist.
Language Independent Code	Typically the same value as the American English version of the display value. See "About the Language Independent Code" on page 378.
Translate	Information-only field used by Siebel Engineering. There is no client or server functionality associated with this field.
Language Name	Name of the language for the Display Value.

For information about fields used for multilingual lists of values, such as Translate, Multilingual, and Language-Independent Code, see "About Multilingual Lists of Values" on page 378 and "Important Fields in List of Values Administration Views" on page 397.

4 Configure a picklist to display the LOV Type.

For more information about configuring picklists, see "Configuring Picklists and Pick Applets" on page 343.

5 Compile and test.

NOTE: To see the additions or changes to a list of values at run, you need to clear the cache.

About Organization Enabled Lists of Values

You can set up LOVs to appear for some organizations but not others. For example, suppose your company has several subsidiary companies and each subsidiary is set up as an organization in your Siebel implementation. For a given picklist, you can display different LOVs to members of each organization. You do this by associating LOVs to specific organizations. See "Associating Organizations to Lists of Values" on page 376 for more information.

When LOVs are associated with an organization, the values displayed to the user may differ depending on whether the user is creating a new record or viewing an existing record.

- If a user is creating a new record, the LOVs associated with the user's current organization (the organization to which the user's active position is associated) are displayed. A user can be associated with multiple organizations if they hold multiple positions, but only the organization of the active position determines which LOVs are displayed.
- If a user is viewing an existing record, the LOVs associated with the record's owner organization are displayed. For example, the organization associated with a user's active position may be *Org ABC*, but the primary organization associated with the record that the user is viewing could be *Org XYZ*. In this case, the LOVs displayed for the picklist would be those associated with *Org XYZ*.

For more information about organizations and access control, see the *Security Guide for Siebel eBusiness Applications*.

Guidelines for Setting Up Organization Enabled Lists of Values

Before setting up organization-enabled LOVs consider the following:

- Organization-enabled LOVs can become complex without careful planning. Be sure to identify all LOV types that require organization-specific LOVs. For each of these LOV types:
  - Define default LOVs (values intended for all organizations that do not require custom LOVs).
  - Define the organizations that require custom LOVs.
  - For each organization, define the custom LOVs for the organization.

- If you require the same value to appear for more than one organization associated with the LOV type, you must create duplicate values for each organization.

  A list of values associated with an organization is displayed to members of that organization only. LOVs not associated with an organization are displayed to all organizations, except the organizations associated with the LOV type. For example, *Value 1* may be associated to *Org ABC*. *Value 2* may be associated to *Org XYZ*. *Value 3* may not be associated to any organization. Value 3 is displayed to all organizations, except *Org ABC* or *Org XYZ*. For *Value 3* to also appear for *Org ABC* and *Org XYZ*, you must create duplicate values and add them to the organization-specific lists of values, one assigned to *Org ABC* and one assigned to *Org XYZ*.

- For large implementations, consider using EIM to load organization-specific LOV data. Be sure to associate the appropriate organizations with the LOV types and a single organization with each LOV value.

  For more information about using EIM, see the *Siebel Enterprise Integration Manager Administration Guide*.

- Organization-enabled LOVs are specific to one organization only. There is no inheritance based on organization hierarchy. For example, an LOV associated with a parent organization does not mean that all child organizations inherit access to the LOV. You must explicitly associate LOVs to each organization.

- After an upgrade be sure to review your LOVs to verify that any LOVs delivered as seed data do not interfere with your custom LOV implementation.

- If you are using both organization-enabled LOVs and MLOVs, be sure one of the following is true:

  - The values for both the Language Independent Code and the Display Value are distinct from all other records.

  - The values for both the Language Independent Code and the Display Value are the same as another record belonging to another organization.

Guidelines for Configuration and Scripting With Organization Enabled LOVs

When configuring objects in the Siebel repository or scripting, consider the following:

- If LookupValue() or LookupName() are used as expressions in repository configuration, be aware that in contexts where there is no data, such as a new record, the organization associated with the user's current position is used to drive visibility and thus the display of organization-enabled LOVs. In contexts where data exists, the primary organization associated with the record is used to drive visibility.

- If LookupValue() or LookupName() are used as functions in repository configuration or scripting, be aware that the organization associated to the users primary position is used to drive visibility and thus the display of organization-enabled LOVs.

- When a user selects an existing record, the organization context is determined by the primary organization associated with the record, not the organization associated with the user's position. The Owner Organization Specifier property on the business component's underlying base table specifies the column that holds the organization Id. (For business components based on S_PARTY, this property is specified on the primary extension table).

  For most tables in the repository, the Organization Specifier property is set to the column holding the primary organization Id. For example, for S_ORG_EXT, it is set to [BU_ID]. The property could also be defined in several levels, allowing you to configure a child business component so that it inherits the organization context from the row in the parent business component. For example, when a user is creating a child record, the LOVs initially displayed are determined by the organization associated with the user, but when the user performs a query on the child applet, LOVs displayed are determined by the organization inherited from the parent record.

  An example of the Organization Specifier property defined with several levels is [S_TAB_X][S_TAB_COL1][S_TAB1_COL2], where each element is one of the following:

  - A column in the current table.
  - The name of an extension table.
  - The name of a FK column.
  - The name of the column holding the BU_ID.

  NOTE: You can only configure the Organization Specifier property for custom tables. The property is read-only for standard Siebel tables. If you have a requirement to modify the Organization Specifier property for a standard Siebel table, contact Siebel Expert Services.

Associating Organizations to Lists of Values

You associate organizations to LOV types and to LOV values. Before associating organizations to LOVs, be sure to review "Guidelines for Setting Up Organization Enabled Lists of Values" on page 374.

To associate organizations to LOV types and LOV values

1. Navigate to the Administration - Data > LOV Explorer view.

   The List of Values Explorer view displays LOV Types in a tree structure. You can expand each LOV Type to see the LOV values associated with the LOV Type.

2. In the List of Values - Type applet, run a query for the LOV Type that requires organization-specific LOV values.

   The records found are displayed in the List of Values - Type applet and in the LOV explorer.

   NOTE: You can also navigate to the LOV Type in the LOV explorer.

3. Select the LOV Type and then click the Select button in the Organization field.
4. In the Organizations dialog box, select the organizations you want, click Add, and then click OK.

   The organizations you add here will be available to associate to LOV values of this LOV Type.

5 In the LOV explorer, expand the LOV Type and the Values folder underneath it.

The list of values for the selected LOV Type are displayed.

6 In the List of Values applet, create sets of LOV values for each organization.

a In the List of Values applet, click New.

b Type a value for the Display Name and Code and select an organization to associate with the LOV value.

NOTE: Code is typically the same as Display Name.

c Repeat for each LOV value that you want to associate to an organization.

Each LOV value can only be associated with one organization. Therefore, if a given value needs to be associated to more than one organization, you must create a duplicate value for each organization.

If an LOV Value is not associated to an organization, it is available to all organizations, except those organizations that are associated with the LOV Type in Step 4.

7 Choose Menu > Clear Cache.

After clearing the cache, the LOV changes take effect.

About Multilingual Lists of Values

You can configure your Siebel application to display multilingual lists of values (MLOV) in static picklists. This allows you to display values in the active language of the user. It also allows the values selected by a user in one language to be retrieved by users working in other languages.

NOTE: To enable MLOV, the static picklist must be bounded and must not be a hierarchical picklist.

For more information about the user's active language, see *Global Deployment Guide*.

Related Topic

"About Static Picklists" on page 344

About the Language Independent Code

The LOV table contains a Display Value column and a Language Independent Code column. Both monolingual and multilingual lists of values display values from Display Value column to the user. However, after the user selects a value in a picklist, the actual value stored in the database is different for monolingual and multilingual lists of values.

- A monolingual picklist stores the Display Value.
- A multilingual picklist stores the Language Independent Code.

For example consider the values in Table 53. A multilingual picklist would display the Display Value (Mr., Señor, or Herr) depending on the active language of the user, but it would store the value Mr. in the database, because that is the value defined in the Language Independent Code column.

Table 53. Example LIC

Display Value	Language Independent Code
Mr.	Mr.
Señor	Mr.
Herr	Mr.

NOTE: Generally the language-independent code value is the same as the American-English version of a particular selection value.

Storing the value from LIC column rather than the Display Value column allows the data to be stored in a form that users working in other languages are able to retrieve and allows the roll up of data for management reports, regardless of the language of the users who enter the data.

CAUTION: The length of the language-independent code (the value stored in the database) must be equal to the longest display value for the MLOV. If it is not, the display value will be truncated. If the standard column does not meet your requirements and you are using a custom extension column, the column must be VARCHAR and have a maximum length (width) of 30.

Guidelines for Configuring MLOVs

- You cannot have children of a list of values entry active when the parent list of values entry is inactive.
- Hierarchical LOVs do not support MLOVs. If you have a requirement for hierarchical MLOVs, contact Siebel Expert Services for assistance.
- LOV_TYPE should not be enabled for multilingual list of values. It has a single-language entry only.

Process of Enabling MLOVs

To set up an MLOV operation, you must modify the LOV configuration in Siebel Tools, as well as perform administration tasks. For MLOV to be enabled the following conditions must be met:

- The column of the field using the picklist has the Translation Table property set to S_LST_OF_VAL.
- The picklist must be bounded (Lov Bounded Property is set to True).
- The picklist must use the same LOV Type as specified in the Lov Type property of the column.

  NOTE: A picklist's LOV Type should always match the LOV Type of the underlying column (the column on which the picklist's field is based).

To enable MLOVs perform the following tasks.

1. "Identifying Which Columns to Enable" on page 380.
   a. "Making Sure the LOV Type Is Translatable" on page 380.
   b. "Determining If the Picklist Is Bounded" on page 381.
   c. "Reviewing List of Columns That Cannot be MLOV Enabled" on page 382
2. "Configuring the Multilingual List of Values in Siebel Tools" on page 383.
3. "Adding Translated Display Values in Application Administration" on page 384.
4. "Upgrading Existing Data Using the MLOV Upgrade Utility" on page 385.
5. "Recompiling and Deploying" on page 390.

NOTE: Configuration of MLOVs can affect performance, especially when the field on which the picklist is based is used as part of a search or sort. Performance characteristics should be considered and verified in conjunction with configuration of MLOVs.

For more information, see *Performance Tuning Guide*.

Columns storing data that is read by server programs, such as Assignment Manager, Siebel Remote, Siebel Anywhere, or Workflow Manager, require additional configuration. See the following for more information:

- "Configuring Siebel Business Process Designer to Use MLOV-Enabled Fields" on page 392
- "Configuring Siebel Assignment Manager to Use MLOV-Enabled Fields" on page 395

■ "Configuring Siebel Anywhere for Use with MLOV-Enabled Fields" on page 397

Configuring MLOVs may also include changes to the Siebel Visibility Rules. Any reference in a Visibility Rule to an LOV entry for a type you plan to configure for multilingual support must be changed from the Display Value to the language-independent Code. Check the visibility rules for references to any LOV entries as part of your configuration of MLOVs.

NOTE: Custom extension columns can always be MLOV enabled.

Identifying Which Columns to Enable

Not every list of values type can be enabled as multilingual. You need to determine which columns you can enable based on the LOV type. LOV types must meet the following conditions:

- The column must be marked as translatable. See "Making Sure the LOV Type Is Translatable" on page 380.
- The picklist must be bounded. See "Determining If the Picklist Is Bounded" on page 381.
- The column must not be one of the "Reviewing List of Columns That Cannot be MLOV Enabled." See "Reviewing List of Columns That Cannot be MLOV Enabled" on page 382.

NOTE: Do not set up a column for a MLOV unless you are sure that you intend to use that column for your implementation.

Checking for Visibility Rules

This task is part of the "Process of Enabling MLOVs" on page 379.

To check for visibility rules

1. In Siebel Tools, navigate to the Dock Object Visibility Rules view.

   Use the flat screen view to simplify searching.
2. Go to the SQL Statement field and search for literals across all rows that are not null.
3. Examine the results for values that need to be translated.

NOTE: You cannot change visibility rules.

Making Sure the LOV Type Is Translatable

A *translatable* type is a list of values type that can be modified, or translated into additional languages, without affecting the functionality of your application. This is indicated in the Translate setting in the List of Values Administration view in Siebel eBusiness Applications.

If an item is translatable, it can be modified without affecting Siebel eBusiness Applications functionality.

This task is part of "Process of Enabling MLOVs" on page 379.

To determine if an LOV type is translatable

1. Connect to the server database using Siebel eBusiness Applications.
2. From the application-level menu, choose View > Site Map > Application Administration >List of Values.
3. For the LOV type that you are interested in, look in the Translate list column for a check mark.

If you add an LOV type, set this list column according to your configuration. Do not change any existing settings shipped with Siebel eBusiness Applications, because these are set to reflect the Siebel eBusiness Applications configuration. Changing this setting will not allow you to enable an LOV type.

Determining If the Picklist Is Bounded

A *bounded picklist* is a picklist where users must choose from the existing choices and cannot enter their own data. An *unbounded picklist* is a picklist where users may either enter their own data or choose from the existing choices. Only bounded picklists can be configured to be multilingual.

You must verify that the picklist is bounded and that the underlying column has the Lov Bounded property set to True. All columns for a particular LOV type must be bounded. If any of the columns for the LOV type is not bounded, then none of the columns can be set to multilingual for that LOV type.

For example, Table 54 shows the columns for the LOV type AVAILABILITY_STATUS. Although three of the columns are LOV bounded, you would not be able to enable these columns as multilingual, because one column (NEXT_AVAIL_CD) is Unbounded. If you were to run the MLOV Upgrade Utility, you would receive an error message that says the columns are inconsistently bounded. For more information, see "Resuming the MLOV Upgrade Utility When Errors Occur" on page 388.

Table 54. Example of Inconsistently Bounded Columns

Name	LOV Type	LOV Bounded
CURR_AVAIL_CD	AVAILABILITY_STATUS	Y
NEXT_AVAIL_CD	AVAILABILITY_STATUS	Y
CURR_AVAIL_CD	AVAILABILITY_STATUS	Y
NEXT_AVAIL_CD	AVAILABILITY_STATUS	N

You can change the LOV Bounded and LOV Type properties of the column object in the following circumstances:

- For standard columns that are not already assigned to an existing LOV type, you can change the LOV Type and LOV Bounded properties as necessary.
- For standard columns that are already assigned to an existing LOV type and have the LOV Bounded property set to FALSE, you can change the LOV Bounded property to TRUE. This is only supported in the context of enabling MLOVs.

This task is part of "Process of Enabling MLOVs" on page 379.

To determine if a picklist is bounded from the Picklists list

1. Connect to the server database using Siebel Tools.
2. Select the Flat tab in the Object Explorer.
3. Select the Pick List object type.
4. Query the Type Value property for the list of values type you are interested in.

   If the Bounded property is checked, then that item is a bounded picklist.

To determine if a picklist is bounded from the Columns list

1. Connect to the server database using Siebel Tools.
2. Select the Flat tab in the Object Explorer.
3. Select the Column object type.
4. Query the LOV Type property for the list of values type you are interested in.

   If the LOV Bounded property has a check mark, then the picklist for that target column is bounded.

   NOTE: The *Translate* property of the column is only for internal use, and has nothing to do with the configuration of MLOVs.

Reviewing List of Columns That Cannot be MLOV Enabled

There are special cases that should be considered when determining whether an LOV TYPE can be enabled for multilingual display. The columns listed in Table 55 cannot be MLOV enabled.

This task is part of the "Process of Enabling MLOVs" on page 379.

Table 55. Columns That Cannot Be MLOV Enabled

Table	Column
S_AGREE_POSTN	APPR_ROLE_CD
S_CONTACT	PREF_LANG_ID

Table 55. Columns That Cannot Be MLOV Enabled

Table	Column
S_CONTACT_X	ATTRIB_48
S_CS_RUN	STATUS_CD
S_DOC_ORDER	TAX_EXEMPT_REASON
S_ONL_LAYOUT	CONTROL_TYPE_CD
S_ORG_EXT	DIVN_CD
S_ORG_EXT	DIVN_TYPE_CD
S_ORG_EXT_XM	NAME
S_PRI_LST_ITEM	PRI_METH_CD
S_PROD_INT_CRSE	CRSE_TYPE_CD
S_PROD_INT_X	ATTRIB_50
S_PROD_INT_X	ATTRIB_51
S_PROD_INT_X	ATTRIB_53
S_PROJ_ORG	PROJ_ROLE_CD
S_PROJITEM	PROD_AREA_CD
S_PROJITEM	STATUS_CD
S_SRC	SRC_CD
S_SRC	STATUS_CD
S_SRC_EVT	FORMAT_CD
S_SRCH_PROP	NAME

Configuring the Multilingual List of Values in Siebel Tools

After you have determined which columns to enable as multilingual, you configure those columns in Siebel Tools.

List of values types are enabled for multilingual support on a target-column basis. Because a list of values type can be used for different target columns, the Multilingual property must be implemented for all target columns that use the same type.

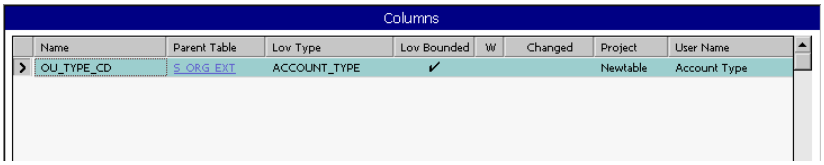
The following procedure describes the process for manually enabling a column. The list of values type ACCOUNT_TYPE is used as an example.

This task is part of the "Process of Enabling MLOVs" on page 379.

To enable a column for multilingual storage and display

1. Open Siebel Tools, and connect to the server database.
2. Select the Flat tab in the Object Explorer.
3. Select the Column object type.
4. Choose Query > New Query, enter the name of the desired list of values type in the LOV Type property, and then press ENTER to execute the query.

   In this example, you would search for an LOV Type of ACCOUNT_TYPE. The query shows you the columns that use that LOV type, in this case there is only one column named OU_TYPE_CD.



5. Check that the columns using that type have a check mark in the LOV Bounded property.
6. Change the Translation Table Name to S_LST_OF_VAL for all the columns returned by the query.

NOTE: Changing the properties of object definitions directly on the server is a nonstandard practice that is used only for configuration of columns for multilingual storage and display. Under all other circumstances, the correct and safe way to change object definition properties is to check out projects to the local repository, make the desired changes, and check them back in to the server. For more information about checking in and checking out projects, see *Using Siebel Tools*.

Adding Translated Display Values in Application Administration

After you have configured a column to be multilingual, display values must be defined for each language that will be supported.

This task is part of the "Process of Enabling MLOVs" on page 379.

To add translated display values

1. Using the Siebel Web client, connect to the server database.
2. From the application-level menu, choose Site Map > Application Administration > List of Values.
3. Find the list of values type for the enabled target columns.
4. For every language that will be supported, create a new record for each display value for that list of values type.

   For instance, if you plan to support German and French in addition to the existing English display values, create two new records for each display value—one in German and one in French.

5 For each new record, the language-independent code must be the same as for the original record, but the entries in the Language and Display Value list columns are set differently, as appropriate.

For more information on adding and maintaining translated values, refer to "Important Fields in List of Values Administration Views" on page 397.

Upgrading Existing Data Using the MLOV Upgrade Utility

After you have configured your application for use with MLOVs and added new display values for all the languages you intend to support, you must upgrade your existing LOV data. You do this using the MLOV Upgrade Utility.

NOTE: Even if you have just completed a new installation of your Siebel application, you must perform this data upgrade.

You run the MLOV Upgrade Utility in two modes.

- **Validation.** Running the utility in this mode validates the current repository for data inconsistencies. If the utility finds inconsistencies, the program stops and writes the errors to a log file.
- **Translation.** Running the utility in this mode:
  - Changes data in target columns that are configured for MLOVs from the display value to the language-independent code.
  - When you set the target column for an LOV Type to multilingual, the utility sets the MULTILINGUAL flag in the LOV table (S_LST_OF_VAL) to make sure of consistency between the multilingual state of the target column and its corresponding List of Values in the LOV table (S_LST_OF_VAL).
  - Verifies that all target columns using the desired MLOV type have been enabled.

    NOTE: Target columns are columns that store either the display value or the language-independent code as part of user data.

The MLOV Upgrade Utility upgrades target columns that are marked as bounded and updates list of values types that are not already marked as multilingual. You can run the utility as often as you need to; only data that has not already been upgraded will be affected.

NOTE: The upgrade process run by MLOV Upgrade Utility is not reversible. Consider performing a backup of the Siebel database prior to running the utility.

You run the MLOV Upgrade Utility using the Siebel Software Configuration Utility. The Siebel Software Configuration Utility is a wizard that will help you define the required parameters for running the MLOV Upgrade Utility. You run the utility in validation mode first, fix errors as they appear, and then run it in translate mode, which will enable your existing data for MLOVs.

NOTE: Before running the MLOV upgrade, drop all indexes from the columns that you are upgrading. After the MLOV upgrade is complete, recreate the indexes.

This task is part of the "Process of Enabling MLOVs" on page 379.

To run the MLOV Upgrade Utility in a Windows Environment

1. Start the Siebel Software Configuration Utility by choosing Start > Siebel Enterprise Server >Configure DB Server.

   NOTE: You can also start the Siebel Software Configuration Utility from the DOS Prompt command line. See "To start the MLOV Upgrade Utility from the DOS prompt" on page 387.

   The Siebel Software Configuration Utility appears.

2. Enter the required parameters to run the MLOV Upgrade Utility in validation Mode.

   For a list of the wizard dialog boxes, options, and required values, see Table 56.

   When you run the MLOV Upgrade Utility, it checks for errors and writes them to a log file. The default name of the log file is `mlovupgd_verify.log` and the default location is the `siebsrvr\LOG` directory.

3. Review the log file and resolve errors as necessary.

   For more information, see "Resuming the MLOV Upgrade Utility When Errors Occur" on page 388.

4. If an error is detected, resume running MLOV Upgrade Utility in validation mode by using the DOS Prompt to navigate to the BIN directory of your Siebel Server root directory (`SIEBEL_ROOT\BIN`), and then at the command prompt typing:

   ```
   siebupg /m master_mlov_verify.ucf
   ```

   The MLOV Upgrade Utility resumes running.

5. Repeat Step 1 through Step 4 until no errors are detected.

6. Start the Siebel Software Utility following the steps described in Step 1.

7. Enter the required parameters to run the MLOV Upgrade Utility in translation mode.

   For a list of the wizard dialog boxes, options, and required values, see Table 56 on page 388.

   The MLOV Upgrade Utility enables your existing data for MLOV. For columns configured for MLOVs, the MLOV Upgrade Utility finds LOV values in user data that are not in S_LST_OF_VAL and inserts them into S_LST_OF_VAL as inactive. It changes the display value of bounded columns to the language independent code and sets the value for the Multilingual attribute to true.

To run the MLOV Upgrade Utility in a UNIX environment

1. Start the Siebel Software Configuration Utility doing the following:

   - Navigate to the Siebel root directory and type source siebenv.csh
   - Type setenv `LANGUAGE DISPLAY_LANGUAGE` (where `DISPLAY_LANGUAGE` represents the three letter code for the display language; for example ENU, FRA, DEU, and so on).
   - Type setenv `SIEBEL_ROOT SIEBEL_ROOT` (where `SIEBEL_ROOT` is the name of the directory where you installed the Siebel Server).
   - Type the command `dbsrvr_config.ksh`

   The Siebel Software Configuration Utility appears.

2 Enter the required parameters to run the MLOV Upgrade Utility in validation Mode.

For a list of the wizard dialog boxes, options, and required values, see Table 56.

When you run the MLOV Upgrade Utility, it checks for errors and writes them to a log file. The default name of the log file is `mlovupgd_verify.log` and the default location is the `siebsrvr\LOG` directory.

3 Review the log file and resolve errors as necessary.

For more information, see "To resume running in translate mode, type `srvrupgwiz /m master_mlov_translate.ucf`" on page 389.

4 If an error is detected, resume running MLOV Upgrade Utility in validation mode by navigating to the bin directory of your Siebel Server root directory (`SIEBEL_ROOT/bin`), and then typing the following command:

```
svrupgwiz /m master_mlov_verify.ucf
```

The MLOV Upgrade Utility resumes running.

5 Repeat Step 1 through Step 4 until no errors are detected.

6 Start the Siebel Software Utility following the steps described in Step 1.

7 Enter the required parameters to run the MLOV Upgrade Utility in translation mode.

For a list of the wizard dialog boxes, options, and required values, see Table 56 on page 388.

The MLOV Upgrade Utility enables your existing data for MLOV. For columns configured for MLOVs, the MLOV Upgrade Utility finds LOV values in user data that are not in S_LST_OF_VAL and inserts them into S_LST_OF_VAL as inactive. It changes the display value of bounded columns to the language independent code and sets the value for the Multilingual attribute to true.

To start the MLOV Upgrade Utility from the DOS prompt

1 From the DOS Prompt, navigate to the \BIN directory of your Siebel Server root directory.

For example: `cd siebsrv/BIN`

2 Run the MLOV Upgrade Utility in validation mode by typing the following at the command prompt:

```
ssincfgw -l language_code -v Y
```

where *language_code* is the three-letter code (all capitals) for the language in which you want to display the GUI.

For example, to run the MLOV Upgrade Utility in English, you would type:

```
ssincfgw -l ENU -v Y
```

The Open dialog box appears.

3 Select dbsrvr.scm and then click Open.

The Siebel Software Configuration Utility -DB Server Configuration dialog box appears.

MLOV Upgrade Utility Parameters

To run the MLOV Upgrade Utility, complete the dialog boxes listed in Table 56 and enter or select the values as you go.

Table 56. MLOV Upgrade Utility

In This Dialog Box	Enter Or Select The Following
Siebel Enterprise Parameters: Gateway Name Server Address	Gateway Name Server Address Enterprise Server Address
Installation and Configuration Parameters: Siebel Server Directory	Siebel Server Directory
Installation and Configuration Parameters: Siebel Database Server Directory	Database Server Directory
Database Server Options: Siebel Database Operation	Run Database Utilities
Database Utilities: Database Utility Selection	Multilingual List of Values Conversion
MLOV Parameters: MLOV Operation	Validate or Translate, depending on the mode you want to run.
Installation and Configuration Parameters: Language Selection	Base language of your Siebel application.
Installation and Configuration Parameters: RDBMS Platform	RDBMS Platform
Installation and Configuration Parameters: OBDC Data Source Name	OBDC Data Source Name
Installation and Configuration Parameters: Database User Name	Database User Name Database Password
Installation and Configuration Parameters: Table Owner	Table Owner Name Table Owner Password
MLOV Parameters: Repository Name	Repository Name
Configuration Parameter Review	Review the parameters you have defined and then click Finish.

Resuming the MLOV Upgrade Utility When Errors Occur

In case of an error, you can resume running the MLOV Upgrade Utility in validation mode or translation mode.

To resume the MLOV Upgrade Utility in a Windows environment

1. At the DOS prompt, navigate to the BIN directory of your Siebel Server root directory (`SIEBEL_ROOT\BIN`).
2. At the command prompt do one of the following:
   - To resume running in validation mode, type siebupg `/m master_mlov_verify.ucf`
   - To resume running in translation mode, type siebupg `/m master_mlov_translate.ucf`

To resume the MLOV Upgrade Utility in a UNIX environment

1. Navigate to the bin directory of your Siebel Server root directory (*`SIEBEL_ROOT`*`/bin`).
2. At the prompt, do one of the following:
   - To resume running in validation mode, type `srvrupgwiz /m master_mlov_verify.ucf`
   - To resume running in translate mode, type `srvrupgwiz /m master_mlov_translate.ucf`

About the MLOV Upgrade Log File

After the utility runs in either validation mode or translation mode, it writes any errors to a log file. The default names of the log files are `mlovupgd_verify.log` and `mlovupgd_translate.log`. The files are located in the `siebsrvr/LOG` directory.

- **LOVs Inconsistently Bounded or Translation Table Property Not Set to S_LST_VAL.** The message that appears in the log file for LOVs that have the bounded property on columns where they are used set inconsistently (one bounded and one not bounded) or LOV domains that do not have the Translation Table property set to S_LST_VAL is the following:

  ```
  The following Validation checks for:

  1- Two or more columns defined in the same LOV domain are inconsistently bounded
  (one bounded, one not)

  2- Two or more columns are defined in the same LOV domain and at least one of them
  does not have a Translation Table Name of  S_LST_OF_VAL.
  ```

  Any errors of these types are listed in the log file. The information listed includes the LOV Type, Column, and Table.

To fix the LOV types that appear in the log file

1. Open Siebel Tools, and connect to the server database.
2. Select the Flat tab in the Object Explorer.
3. Select the Column object type.
4. Choose Query > New Query, and then enter the name of the list of values type that has a problem in the LOV Type property.
5. Press Enter to execute the query.

6 For the columns displayed, make sure they are LOV BOUNDED = Y.

7 Set the Translation Table Name for the columns displayed to S_LST_OF_VAL.

8 Run MLOV Upgrade Utility in validation mode to make sure that there are no more errors.

- **LOV Domains Not in the S_LST_OF_VAL Table.** The message that appears in the log file for LOV domains that are not represented in S_LST_OF_VAL table is the following:

```
The following validation checks for:

LOV domains in the repository that are not represented in S_LST_OF_VAL
```

  This message means that an LOV domain is in the repository, but is not represented as a value in the list of values table, with a list of values type of LOV_TYPE. This can happen when you delete a record in the list of values table, instead of deactivating it, or when you enter an incorrect entry in the LOV Type property for a column added using a database extension.

  For more information, refer to "Deleting Compared to Deactivating MLOV Records" on page 398.

  To fix this problem, add the LOV domain in the List of Values Administration view and specify LOV_TYPE in the Type list column, or correct the entry in the LOV Type property in the repository. For more information, see "About Adding Records for MLOV Fields" on page 398.

  For any values found in the target tables without matching records in the list of values table, the script will create a matching record in the list of values table. These records are marked as inactive. Remember to add language-specific entries for these base records, so that they display in the active language.

Recompiling and Deploying

Every time you change the configuration to enable another column to be multilingual, you must compile a new .srf file based on the newly configured repository. Only the Newtable project needs to be compiled again. Additionally, you need to deploy the changes to users so that users can see the configured picklists in the desired language.

Integration Considerations

Enabling MLOVs does not affect just the Siebel eBusiness Applications client and the relevant target tables. Other features in your Siebel eBusiness Applications implementation must also consider this new configuration.

With Enterprise Integration Manager (EIM), you can import and export data. You can import data into both the list of values table and other tables in Siebel eBusiness Applications.

When importing data into the list of values table, the source table must have a language code and a name-value pair. This pair consists of the Display Value and the Language Independent Code.

When importing data into any other table, you must provide a language code for the /LANGUAGE command-line parameter for EIM. The source table must include the display value for multilingual columns in the language specified in the parameter. EIM validates imported data against list of values entries. The incoming data will be converted to associated language-independent codes during the import.

List of values entries that are marked inactive are ignored during the validation of multilingual LOV values during import by EIM.

When exporting data, you must specify a language code for the /LANGUAGE parameter, so that EIM can correctly translate the language-independent code in the table to the display value during the export.

For more information about command-line parameters for EIM, and information on EIM in general, see *Siebel Enterprise Integration Manager Administration Guide*.

Configuration Considerations

MLOVs are implemented below the business component level, so there are no special configuration considerations, other than what is described here. Fields that point to MLOVs with enabled target columns will automatically return display values that match the client language setting.

Because MLOVs are configured on a column basis, target columns that are not configured to be multilingual will behave as before; that is, target columns will store display values instead of language-independent codes.

For display, the underlying language-independent code is converted to its corresponding display value using a Siebel eBusiness Applications lookup. For searching and sorting, however, a database join is performed by your Siebel application to the list of values table. Therefore, when configuring the application, make sure that any configuration directly involving the list of values table is compatible with your Siebel application MLOV functionality.

Only one multilingual picklist type for each column. This means that for a table that has more than one business component mapped to it and several fields mapped to the same column, a multilingual LOV can be associated with only one of the fields. The MLOV upgrade utility checks for this when running in validation mode. See "Upgrading Existing Data Using the MLOV Upgrade Utility" on page 385.

MLOV Configuration and Coding Guidelines

The following guidelines should be followed when MLOVs are enabled in your environment.

- **LookupName and LookUpValue functions.** These functions can only be used in calculated fields or search specification expressions. They cannot be used with Siebel scripting.
- **Pre/Post default values for fields with LOV picklists.** Always use the LookupValue function with *Expr:* in front of it. The first argument is the LOV Type and the second is the LIC. The function returns the language-specific Display Value. For example:

  ```
  Expr: "LookupValue (""FS_PROD_ALLOC_RULES"", ""Default"")"
  ```

- **Dynamic drilldowns and toggle applets.** These are usually based on a field that has an LOV value. For example, a dynamic drilldown might navigate the user to a Credit Card screen if the account type is equal to *Credit Card* or to a Savings screen if the account type is equal to *Savings*. Do not hard-code the drilldown or toggle conditions. Rather, use the LookupValue function (as described in the previous bullet).

- **Search specs for business components, links, applets, and picklists.** Always use the LookupValue function. For example:

  ```
  [Invoice Code] = LookupValue('FS_INVOICE_CODE', 'Auction')
  ```

- **VB functionality.** VB does not offer a function to retrieve the language-specific Display Value. However, the Display Value must never be hard-coded; you should use the language-independent code instead. To write VB code using the language-independent code only, you must create calculated fields that hold the language-specific translation for a language-independent code.
- **Language and ResourceLanguage parameters.** Set these parameters *only* in the configuration file, for example, Language =<lang>, ResourceLanguage = ENU. If you do not set these parameters only in the configuration file, for example, when /L=<language>and ResourceLanguage = ENU, you will intermittently receive error 2009.

About Querying and Multilingual Lists of Values

To run queries against fields that are controlled by MLOVs, use the Display Value for the search specification; do not use the Language Independent Code for querying. Querying will translate the search specification to the appropriate Language Independent Code to perform the query.

The Display Value used as the search specification should correspond to the Language being used by the application performing the query. If the query is being run through one of the Siebel interfaces (such as CORBA or COM), then the Language used for this translation is specified in the .cfg file used with the interface.

There is no difference to the user in the apparent functionality of the product when MLOVs are on or off. Internally, searches are applied using a function applied to the language-independent code. You can also do this with predefined queries and search expressions in the repository by using the same function (LookupValue (LOV Type, Language-Independent Code)). For more information about the LookupValue function and how it is used with MLOVs, see "MLOV Configuration and Coding Guidelines" on page 391.

For more information about Query Operators and Expressions, see *Siebel Developer's Reference*.

Configuring Siebel Business Process Designer to Use MLOV-Enabled Fields

Additional configuration is required to enable the Siebel Business Process Designer to use MLOV-enabled fields. Siebel Business Process Designer compares values in target tables with values in the Business Process administration tables to determine whether a particular condition is true. For columns enabled for MLOVs, the value stored in the target table is the Language-Independent Code rather than the Display Value. However, the value in the Business Process Administration table is the Display Value. Siebel Business Process Designer cannot evaluate a condition by comparing the Language-Independent Code to the Display Value.

To enable Siebel Business Process Designer to work with MLOV-enabled columns, you must configure Workflow entities so that they compare the language-independent code in the target table with the language-independent code in the Business Process Designer administration table. You must do this for the following entities:

- "Policy Conditions" on page 393
- "Action Arguments" on page 394

Policy Conditions

Before you enable Policy Conditions, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV-enabled, identify which ones are referenced by Policy conditions.

For each of the fields that reference a Workflow Policy condition, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Workflow Policy Column to use the new picklist and applet.
- Repick the values for existing workflow policies.

To create a LIC picklist for a Workflow Policy Column

1. In Siebel Tools, navigate to the Workflow Policy Column object type that you want to enable to use with MLOVs.
2. Find the Workflow Policy Column that references the MLOV enabled field.
3. In the PickList property field, click the picklist name.

   The Pick Lists window appears in the Object List Editor.
4. Create a new picklist by copying the existing one and append LIC to the name.

   For example, Picklist Account Status LIC.
5. Verify that the Sort Specification property of the picklist is set to "Name."

To create a new LIC applet for a Workflow Policy Column

1. Navigate back to the Workflow Policy Column selected in the previous procedure.
2. In the Applet property field, double-click the name of the associated applet.

   The Applet window appears in the Object List Editor.
3. Create a new applet by copying the existing one and append LIC to the name.
4. Add a new list column to the applet for the language-independent Code.
   a. In the Object List Editor, select List object type and then select the List Column object type.

b In the List Column window, create a new record by copying an existing one, and then set the Field property to "Name."

To configure the Workflow Policy Column

1 Navigate back to the Workflow Policy Column selected in the previous section.

2 In the PickList property field, select the new picklist created in Step 4 of the procedure "To create a LIC picklist for a Workflow Policy Column" on page 393.

3 In the Source Field property, change the value from Value to Name.

4 Compile changes.

To repick the values

1 Log on using a client connected to the modified repository file.

2 From the application level menu, choose View > Site Map > Business Process Administration > Workflow Policies.

3 Repick the Values by selecting the conditions and reselecting the appropriate display values.

This will store the language-independent code.

Action Arguments

Before you enable Action Arguments, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV enabled, identify which ones are referenced by Policy conditions.

For each of the fields that reference an Action Argument, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Action Arguments to use the new picklist and applet.
- Repick the values for the existing workflow policies.

To create a LIC picklist for a Workflow Policy Program Argument

1 In Siebel Tools, navigate to the Workflow Policy Program object type and the Workflow Policy Program that contains the argument that you want to enable for use with MLOVs.

2 Select the Workflow Policy Program Argument object type (child of Workflow Policy Program) and then select the Argument you want to enable for use with MLOVs.

3 In the PickList property field, click the picklist name.

The PickLists window appears in the Object List Editor.

4 Create a new picklist by copying the existing one and append LIC to the name.

To create a new LIC applet for a Workflow Policy Program Argument

1. Navigate back to the Workflow Policy Program Argument selected in the previous procedure.
2. In the Applet property field, double-click the name of the associated applet.

   If no applet exists, you must create one.

   The Applet window appears in the Object List Editor.
3. Create a new applet by copying the existing one and append LIC to the name.
4. Add a new list column to the applet for the language-independent code.
   a. In the Object List Editor, select List object type and then select the List Column object type.
   b. In the List Column window, create a new record by copying an existing one and set the Field property to Name.

To configure the Workflow Policy Program Argument

1. Navigate back to the Workflow Policy Program Argument selected in the previous section.
2. In the PickList property field, select the new picklist created in Step 4 of the procedure "To create a LIC picklist for a Workflow Policy Program Argument" on page 394.
3. In the Source Field property, change the value from Value to Name.
4. Compile changes.

To repick the values

1. Log on using a client connected to the modified repository file.
2. From the application level menu, choose View > Site Map > Business Process Administration > Workflow Policies.
3. Repick the Values of arguments for existing workflow policies.

For more information about Siebel Business Process Designer, see *Siebel Business Process Designer Administration Guide.*

Configuring Siebel Assignment Manager to Use MLOV-Enabled Fields

Additional configuration is required to enable Siebel Assignment Manage to use MLOV-enabled fields. Siebel Assignment Manager compares values in target tables with values in Assignment Manager administration tables to determine whether a particular condition is true. For columns enabled for MLOVs, the value stored in the target table is the Language-Independent Code rather than the Display Value. However, the value in the Assignment Manager administration table is the Display Value. Assignment Manager cannot evaluate a condition by comparing the Language-Independent Code to the Display Value.

To enable Siebel Assignment Manager to work with MLOV-enabled columns, you must configure Assignment Manager entities so that they compare the language-independent code in the target table with the language-independent code in the Assignment Manager administration table. You must do this for the following entities:

- "Criteria Values and Criteria Skills" on page 396
- "Workload Rules" on page 396

Criteria Values and Criteria Skills

Before configuring Criteria Values and Criteria Skills, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV enabled, identify which ones are referenced by Criteria Values or Criteria Skills.

For each of the fields that reference Criteria Values or Criteria Skills (Assignment Attributes), you must set the Translate column to True and define the language-independent code field as the Translate Pick Field.

To configure Assignment Criteria and Skills for MLOVs

1. In the Object Explorer, select the Assignment Attribute object type.
2. In the Object List Editor, select the Assignment Attribute that you want to work with MLOV-enabled fields.
3. Set the Translate property for the Assignment Attribute to TRUE.
4. Set the Translate Pick Field property to the field name that stores the language-independent code.

   Typically the Name field stores the language-independent code.
5. Compile changes.

Workload Rules

Before configuring Workload Rules, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV-enabled, identify which ones are referenced by Workload Rules.

For each of the fields that reference Workload rules, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Workflow Policy Column to use the new picklist and applet.
- Repick the values for existing records.

The detailed steps for completing these tasks are the same as the steps for configuring Workflow Policy Columns covered in "Policy Conditions" on page 393.

For more information about Siebel Assignment Manager, see *Siebel Assignment Manager Administration Guide*.

Configuring Siebel Anywhere for Use with MLOV-Enabled Fields

Siebel Anywhere requires additional configuration to be able to use fields enabled for MLOVs.

To configure Siebel Anywhere for MLOVs

1. Open Siebel Tools.
2. In the Object Explorer, select the Table object type.
3. In the Tables window, query for S_UPG_KIT.
4. In the Object Explorer, select Column object type (child of Table).
5. In the Column window, select the STATUS column.
6. In the Translation Table Name field, click the drop-down list and select the S_LST_OF_VAL.
7. Compile an .srf file.

   This enables Siebel Anywhere to use MLOV-enabled fields.

After completing this procedure, you can perform standard tasks associated with Siebel Anywhere, such as creating a client repository upgrade kit and distributing to clients.

- Distribute the upgrade kit to Mobile Web Clients.
- Upgrade Siebel Servers with the new .srf file.

NOTE: Distributing Siebel executable to multilingual remote clients requires additional configuration. See *Siebel Anywhere Administration Guide*.

Important Fields in List of Values Administration Views

Several list columns in the list of values views help you to administer multilingual LOVs:

- **Multilingual.** This field indicates which list of values types have been configured to be multilingual. When you run the MLOV upgrade utility (mlovupgd.exe), it sets this flag for the list of values entries. For more information on the upgrade script, refer to "Upgrading Existing Data Using the MLOV Upgrade Utility" on page 385.

  If you add entries after the script has been executed, you must manually update this information to reflect your configuration.

- **Language.** This field indicates for which language the entry is valid. The entries for this picklist come from the Language Administration view. To access this view, from the application-level menu, choose View > Site Map > Application Administration >Languages.
- **Translate.** This field indicates whether the entry's display value can be changed—for instance, translated to another language.

  Only the LOV types that are marked as translatable are candidates for multilingual configuration. For any entries added, you must update this information manually to reflect your configuration. Do not change existing Siebel entries. Doing so will not allow the LOV to be translated.
- **Language-Independent Code (or Code in the Explorer view).** This field is the internal code used for a list of values entry. It is stored in the database when MLOV is enabled and referenced by configurations. The language-independent code must be 30 characters or less. It is typically the English-American version of a particular selection value. The language-independent code cannot be changed.
- **Display Value.** This field is required and holds the text that will appear in picklists. The display value is stored in the database when MLOV is not enabled.

  If there are display values for more than one language for a list of values entry, the display value shown is determined by the current active language.

About Adding Records for MLOV Fields

When you add a new list of values record for a LOV type that has been multilingual-enabled, you also need to add records for all supported languages. For example, when adding a new entry for FREIGHT-TERMS type, you need to add values for supported languages.

If you add a new entry and do not add corresponding additional records for each supported language, the language-independent code will be displayed instead of the display value when a user with one of these languages tries to view the information.

Adding records for all the languages you support is also important for Assignment Manager. For more information, refer to "Configuration Considerations" on page 391.

For more information about adding records to the LOV table, see *Applications Administration Guide*.

Deleting Compared to Deactivating MLOV Records

As you administer MLOVs, you may find that there are records that you no longer need and would like to make inactive. If you delete a MLOV record, then records in other tables that have already been entered using that list of values record will no longer display correctly. The display value in the list of values entries is used to display the language-specific text.

Instead of deleting a record, inactivate it. Inactive values that have already been used and are referenced in other tables in the database will still display correctly. Inactive records are not included in any picklists, however, and are ignored by EIM when it performs validation against LOVs.

The Active list column in the List of Values Administration view is checked by default. To deactivate a record, remove the check mark.

If you try to delete a record, you will get a message asking you if you really want to delete the record, or just deactivate it. If you choose Inactivate from the dialog box, the check mark in the Active list column is removed.

If you do delete an LOV record, any language-independent codes in the target columns referring to the deleted record will then display the language-independent code. Searching and sorting will not function correctly on these values.

19 Configuring Multi-Value Group and Association Applets

Topics in This Chapter

About MVG Applets

A *multi-value group (MVG) applet* (shown in Figure 85) is a dialog box that provides the means to display and maintain a set of records of data from another business component associated with the currently displayed business component record. The multi-value group applet is invoked from a control or list column in the originating applet.

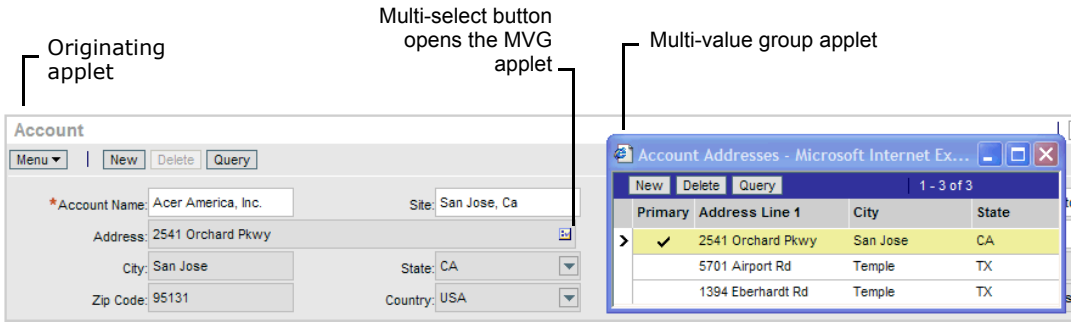


Figure 85. Multi-Value Group Applet

Figure 85 shows the Account Addresses multi-value group applet. It is invoked when the user clicks the check mark button to the right of the Address Line 1 text box. This multi-value group applet lists one or more addresses for the account. The record with a check mark in the Primary check box is the one whose data appears in corresponding controls in the originating applet. While the multi-value group applet is open, the user can view the entire list of team member records for this account, not just the primary one. The user can also add, query, and delete records in this window.

Understanding How MVGs are Implemented

This section summarizes how MVGs are implemented. It covers the following topics:

Multi-value group applets are implemented using object types illustrated in Figure 86.

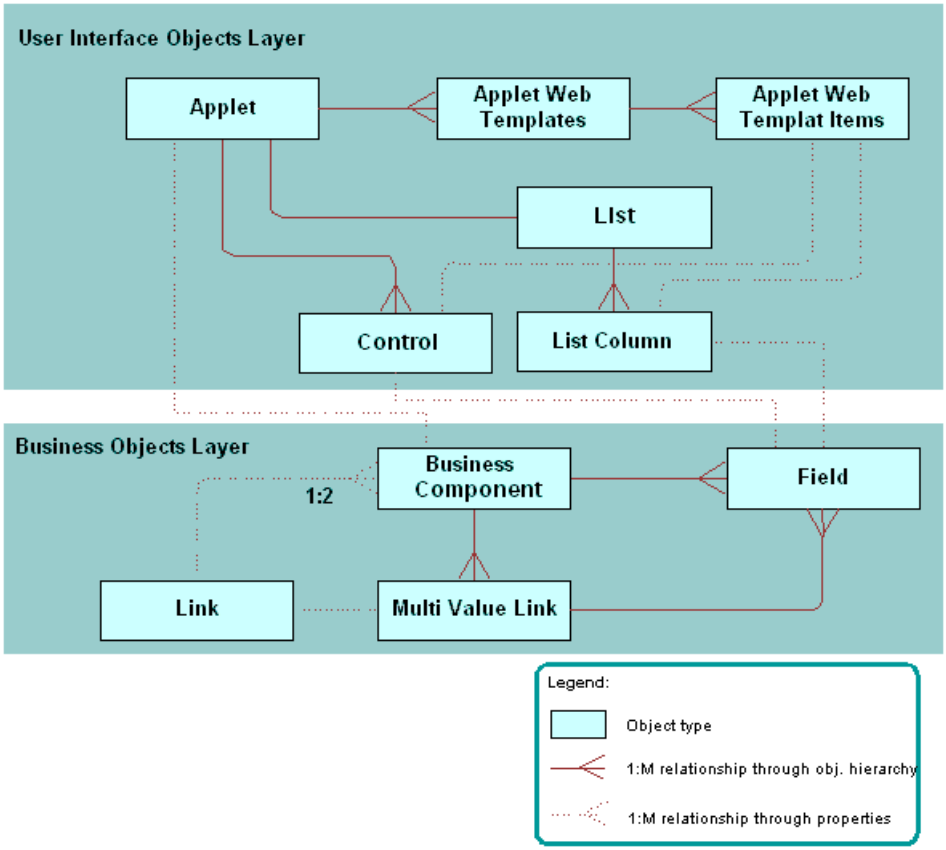


Figure 86. Multi-Value Group Architecture

Figure 87 shows the object definitions used in the implementation of a multi-value group applet in greater detail, and identifies their interrelationships.

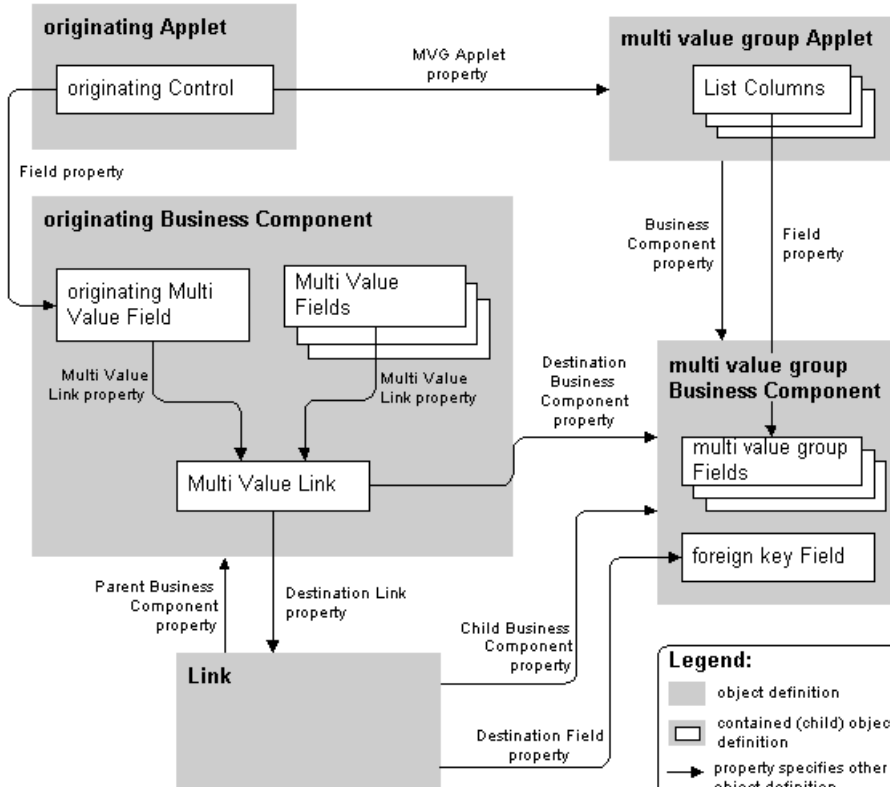


Figure 87. Multi-Value Group Details

The roles of the object definitions in Figure 87 are summarized in the following list and discussed in greater detail in the subsequent sections. The multi-value group example refers to the Account Address MVG applet illustrated in Figure 85 on page 401.

Each of the following objects is discussed in greater detail in the following sections.

- **Originating applet.** Contains the control or list column that invokes the multi-value group applet. In the example, the originating applet is called Account Entry applet.
- **Originating business component.** Business component of the originating applet. This business component (in the example, the Account business component) supplies the data presented in the originating applet (Account Entry applet).

- **Multi-value fields.** Fields in the originating business component that are populated by the multi-value link. Data population relationship is indicated by the presence of the multi-value link's name in their multi-value link property. The Field property in each identifies the corresponding field in the multi-value group business component that provides its data. Multi-value fields used in this multi-value group situation are Street Address, Address Id, City, Country, Fax Number, Postal Code, and State.

  NOTE: If the field is a multi-value field, then the Required attribute will be ignored. In this case you can use a script in Siebel VB or Siebel eScript or the primary address field, if it exists.

- **Multi-value links.** Child of the originating business component. It identifies the link that provides the field values from the multi-value group business component. In the example, the multi-value link is Business Address.
- **Links.** Specifies the master-detail relationship between the originating and multi-value group business components. This is a property of the Multi Value Link object definition from which the fields in the originating business component obtain their values. In the example, the link is Account/Business Address.
- **Multi-value group applet.** Dialog box that appears when the user clicks on the ellipsis button in the originating applet. It lists the multi-value group business component records that are detail records in the master-detail relationship with the current originating business component record. It also provides the means to add, edit, and delete detail records. In the example, the multi-value group applet is called Account Address MVG Applet.
- **Multi-value group business component.** Stores the detail records of the master-detail relationship with the originating business component. The records displayed in the multi-value group applet are those in the multi-value group business component. In the example, the multi-value group business component is called Business Address.

About the Originating Applet of an MVG

The originating applet contains the control or list column that invokes the multi-value group applet. The originating applet has the following important property:

- **Business Component.** Identifies the originating business component.

The originating Control object or List Column object has the following important properties:

- **Field.** Identifies the originating field in the originating business component.
- **MVG Applet.** Name of the multi-value group applet to be invoked.
- **Runtime.** Must be set to TRUE.

About the Originating Business Component of an MVG

The originating business component is the business component of the originating applet. The data values that appear in the originating field and other multi-value fields are obtained from corresponding fields in a record in the multi-value group business component. The record from which these values are obtained is the one indicated as primary.

The originating business component has no essential properties for the configuration of a multi-value group. However, the field and Multi Value Link child object definitions are significant.

The originating field is the field specified in the Field property of the originating control or list column. Other than its relationship with the originating control, its role is identical to that of the other multi-value fields sharing the multi-value link. (A multi-value field is a field with a non-null Multi Value Link property.) Each of the multi-value fields participating in the multi-value group has the name of the multi-value link in its Multi Value Link property. The multi-value fields in the originating business component have the following important properties:

- **Multi Value Link.** Identifies the multi-value link that provides values, by way of the link object definition, from the multi-value group business component.
- **Field.** Identifies the field in the multi-value group business component that, by way of the multi-value link and link object definitions, provides values for the field in the originating business component.

Pick maps can be used for multi-value fields similarly to how they are used for single-value fields. The MVF Pick Map object is a child object of Multi Value Field. Each pick map defines a correspondence between a field in the multi-value group business component and one in the originating business component. These correspondences provide the information required to immediately update the current originating business component record with information from the MVG business component when a record is picked.

Each MVF Pick Map object definition has two important properties:

- **Field.** Identifies a field in the originating business component that is to be populated by data from a field in the MVG business component, when the PickRecord method is invoked.
- **Pick List Field.** Identifies a field in the MVG business component that is the source of data for the field in the Field property of the Pick Map object.

An example is the MVF pick map on the State multi-value field of the Account business component. Account has a multi-value link to the Business Address business component, where it obtains address information.

About the Multi-Value Link

The multi-value link object definition is a child of the originating business component. It identifies the link that provides the field values from the multi-value group business component. The multi-value link in the originating or joined (in the case of an indirect multi-value link) business component has the following important properties:

- **Destination Link.** Identifies the link object definition that provides the master-detail relationship between the originating (or joined) and multi-value group business components.
- **Primary Id Field.** Identifies the foreign key field in the originating, or joined, business component. The foreign key field identifies the primary record in the set of records for one multi-value group (in the multi-value group business component). The primary record is the one that displays in the originating or employing business component.

- **Destination Business Component.** Name of the child business component.

NOTE: An indirect multi-value link may be used in place of a conventional multi-value link when there is an existing link object definition which would be appropriate for use in a multi-value link, but the originating business component is different from the master business component. If there is a Join object definition that joins the desired master business component to the master business component of the link, the existing link can be used in the multi-value link.

For more information about multi-value links, see "About Multi-Value Links" on page 211.

About the Link Object

The Link object definition specifies the master-detail relationship between the originating and multi-value group business components. This makes possible the Link object definition from which the fields in the originating business component obtain their values. The Link object type has the following important properties:

- **Parent business component.** Identifies the originating business component.
- **Child business component.** Identifies the multi-value group business component.
- **Source Field.** Identifies the field in the originating business component that serves as a unique ID to that business component. If this property is blank, it indicates that the field that maps to the ROW_ID column, generally called Id, is the source field.
- **Destination Field.** Identifies the field in the multi-value group business component that identifies the master record for each detail record. It is a foreign key that points back to the originating business component.

  NOTE: In a link based on an intersection table (the relationship between the originating and MVG business component is many-to-many—that is, one in which the Inter Table, Inter Parent Column and Inter Child Column properties are nonblank), you do not specify the Source Field or Destination Field properties. You can specify a source field (although it is not common to do so). Destination field always defaults to Id.

For more information about links, see "About Links" on page 206.

About the MVG Business Component

This business component stores the detail records of the master-detail relationship with the originating business component. The records displayed in the multi-value-group applet are from the multi-value group business component.

The multi-value group business component has no important properties with respect to its role in the implementation of a multi-value group. It has field child object definitions that are used in the following ways:

- To store data for a field in the multi-value group

  Each field with this role is represented by a list column in the multi-value group applet. It may also participate in the multi-value link to supply data to a corresponding field in the originating business component.

- To identify the primary record in the multi-value group

  Primary records are identified by the primary field which is specified in the Primary Field Id property in the multi-value link.

  NOTE: The primary field has nothing to do with the multi-value group business component. It does have relevance to the originating business component, the multi-value link, and the multi-value group applet.

- As the destination field of the link

  The field with this role is a foreign key to the originating business component.

Creating MVGs Using the MVG Wizard

You can use the MVG Wizard to help you create the necessary relationships between business components and define multi-value fields.

To configure a multi-value group using the MVG wizard

1. Choose File > New Objects.

   The New Object Wizards dialog box appears.

2. Under the General Tab, double-click the MVG icon.

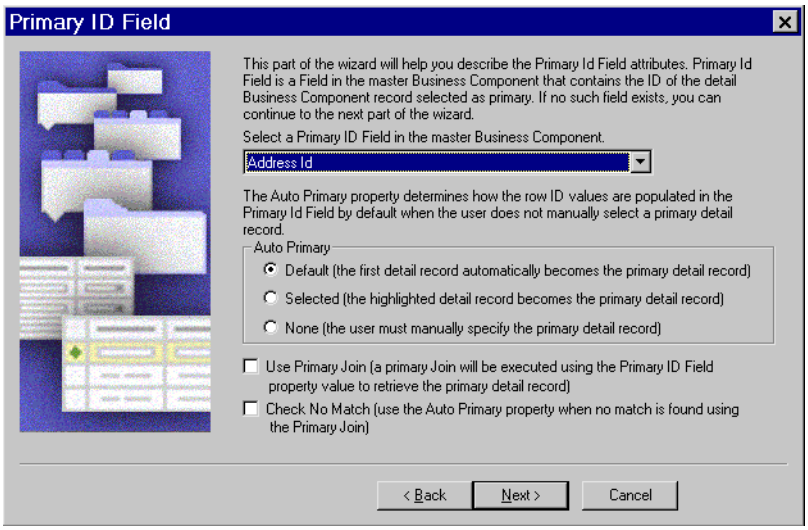
   The Multi Value Group dialog box appears.

3. In the Multi Value Group dialog box, select:
   - The project to which the MVG will belong. Only locked projects are available for you to select.
   - The master business component. The business component must belong to the selected project.
   - Click Next.
4. In the second Multi Value Group dialog box, do the following:
   - Select the detailed business component.
   - Enter a name for the Multi Value Link.
   - Click Next.
5. In the Direct Links dialog box, select the link that you want to use, and then click Next.

   The available links are those that already exist between the master and detail business component.

6. In the Primary ID Field, do the following:

- Select the Primary ID Field in the Master Business Component.
- Set the value for the Auto Primary property.
- Select other options as necessary.
- Click Next.



7 In the Multi Value Link dialog box, select the properties that you want to define and then click Next.

8 In the Multi Value Fields dialog box, enter information for creating multi-value fields on the master business component.

- Select a field on destination business component.
- Enter a name for the multi-value field.
- Click Add.
- Repeat for each field you want to add.
- Click Next.

9 In the Finish dialog box, review the information you have entered for the MVG and then click Finish.

About MVG Applets

MVG applets appear when the user clicks the select icon in the originating applet. It lists the multi-value group business component records that are detail records in the master-detail relationship with the current originating business component record. It also provides the means to add and delete detail records. The multi-value group applet contains list column object definitions that present the data from corresponding fields in the multi-value group business component.

The multi-value-group applet has the following important properties:

- **Business Component.** Identifies the multi-value group business component.
- **Class.** Enter a value of CSSFrameList in this property. This setting indicates that this is a standard list applet.
- **Type.** Enter a value of MVG in this property. This setting indicates that this is a multi-value group applet. This configures the behavior of the dialog box and button controls.
- **Title.** Identifies the multi-value group applet to appear in the title bar.

The List Column object definitions in the multi-value group applet have the following important property:

- **Field.** Identifies the field in the multi-value group business component from which the list column displays data.

  For more information on field data types, see "About Field Data Types" on page 180.

Configuring MVG Applets

You can use the MVG Applet Wizard to create an MVG applet. It will help you define all the necessary object definitions for an MVG applet.

Always name multi-value group applets <BusComp> Mvg Applet. (Note the case sensitivity of Mvg.)

To create an MVG applet

1. Choose File >New Object from the Siebel Tools main menu.

   The New Object Wizard dialog box appears.

2. Click the Applets tab and then double-click the MVG Applet icon.

   The General page of the MVG Applet Wizard appears.

3. In the General Page, enter the following information for the applet, and then click Next.

Field	Comment
Project	Only locked projects appear in the picklist.
Business Component	The business component that the applet is based on.
Applet Name	A unique name for the Applet.
Display Title	The name to appear in the user interface.

   The wizard will use information to create an applet object and define the required applet properties.

4 In the Web Layout-General page, enter the Web templates to use for the applet, and then click next.

Typically, MVG applets use the Popup List template. For a complete description of templates, see *Siebel Developer's Reference*.

MVG applets applets that have base, edit or edit list mode defined, inherited their mode from the parent applet. However, if an MVG applet has only base mode defined, then base mode is always used for the MVG applet regardless of the parent applet's mode.

If you plan to include a New button on your MVG applet, you will also need to define an Edit mode manually, using the Popup Query template.

5 In the Web Layout - Fields page, select the fields that you want to appear on the applet, and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in Step 3 on page 409.

6 In the second Web Layout-Fields page, choose the controls in the Available Controls box that you want to appear on the applet, and then click Next.

All the entries in the Selected Controls box are added by default. If you wish to exclude some of the controls and move them to the Available Controls box, select the controls and click the activated arrow.

NOTE: The available controls come from the "Model HTML Controls" applet. This applet specifies the available controls and also to which template each control is mapped. Users can modify this applet if necessary by adding or removing controls from the applet.

7 Review the information displayed in the Finish page, and then click Finish.

The MVG Applet Wizard creates the applet and supporting object definitions based on the selections you made.

For additional information related to shuttle applets, see *Release Notes* on Siebel SupportWeb.

About Association Applets

A*ssociation applets* are used with pairs of business components that have a many-to-many relationship. Association applets can be invoked from master-detail views or from MVG applets. They allow users to associate a parent record with one or more children through an intersection table.

NOTE: You cannot modify records in an association applet.

Figure 88 shows a many-to-many relationship between two business components.

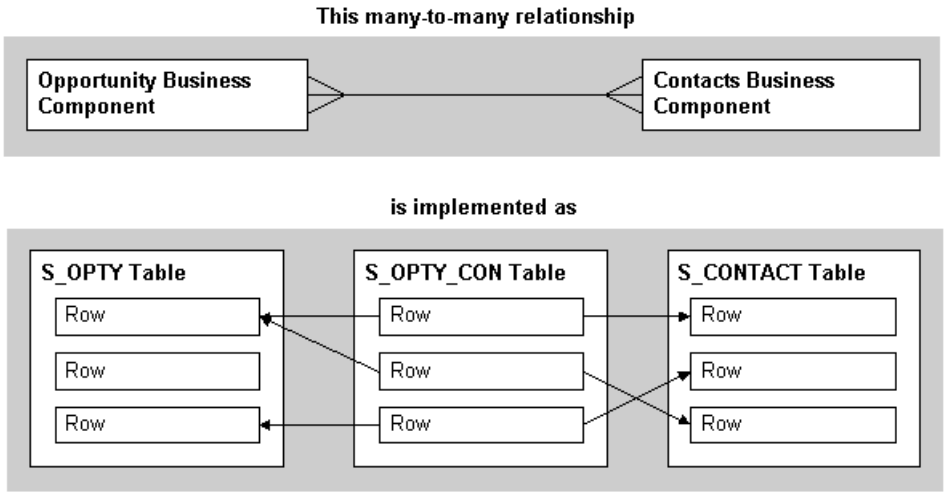


Figure 88. Row Relationships in a Many-to-Many Relationship

"Adding" a record to the detail business component in a many-to-many relationship can in reality mean associating an existing detail record to a master record, rather than creating a new detail record from scratch. This is because master and detail are relative terms in a many-to-many relationship. For example, the Opportunity and Contacts business components in Figure 88 can be displayed as one opportunity to many contacts, or one contact to many opportunities, depending on the active view.

In this situation, the association applet presents the user with a selection list of available detail records. If users see the desired detail record in the selection list, they choose it. If not, they have the option to create a new detail record. In the context of a many-to-many relationship, creating a new association for an existing detail record is called association; creating a new detail record and an association is called addition. Both association and addition of detail records result in the creation of a new row in the intersection table. Addition also results in the creation of a new row in the detail table.

NOTE: An association applet cannot be configured to be constrained or filtered through properties the way a pick applet can (using the Constrain property of a Pick List). To constrain an association applet, you must use Siebel VB or Siebel eScript to query using the Exists clause in the `WebApplet_Load` event on the association applet.

Association Applets Invoked from Master-Detail Views

Association applets are invoked from master detail views where the underlying business components have a many-to-many relationship. The association applet lists the records from a business component. The user selects one or more records using the Find and Starting With controls, if needed, and clicks the OK button to associate the selected record with the active master record.

Figure 89 illustrates an association applet invoked in a master-detail view by selecting Add button in the Contacts list applet.

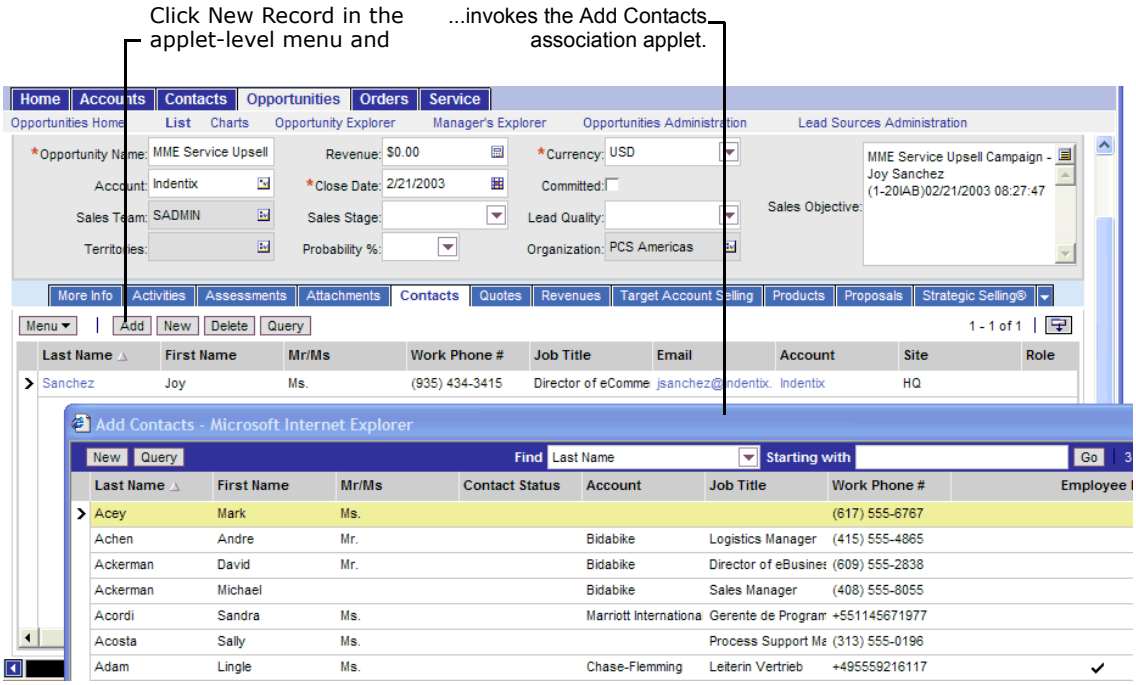


Figure 89. Association Applet Invoked from a Master-Detail View

The master-detail view in Figure 89 is Opportunity Detail - Contacts View, one of two master-detail views displaying opportunity and contact information. The other is Contacts Detail - Opportunities List View, which displays the inverse master-detail relationship. When the user chooses New Record from the Opportunities edit menu, the Opportunities dialog box appears for selection of an existing opportunity record to insert, or for creation of a new opportunity record. A new opportunity record is created by clicking the New button and then entering data into the new record in the Add Opportunities dialog box.

For example, the Add Contacts dialog box is implemented as an association applet called Contact Assoc applet.

Association applets are implemented using the object types illustrated in Figure 90.

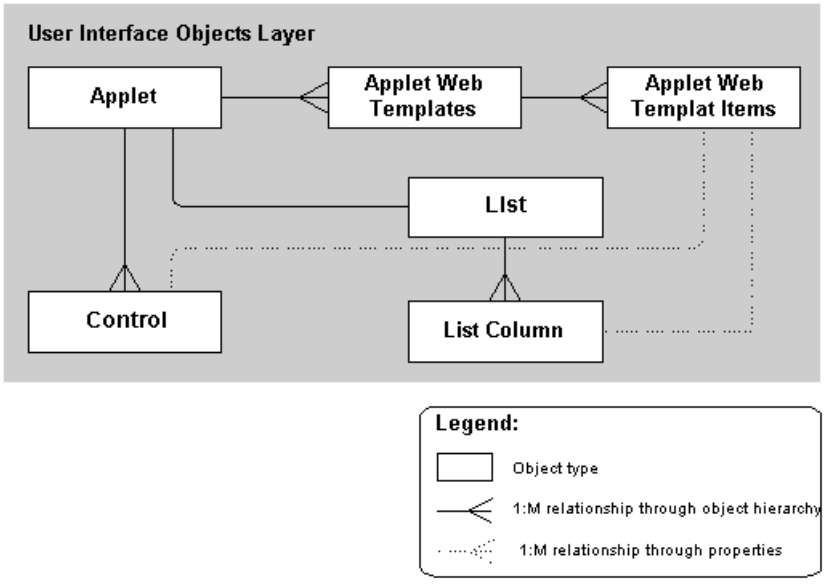


Figure 90. Association Applet Architecture

The details of the object relationships are shown in Figure 91.

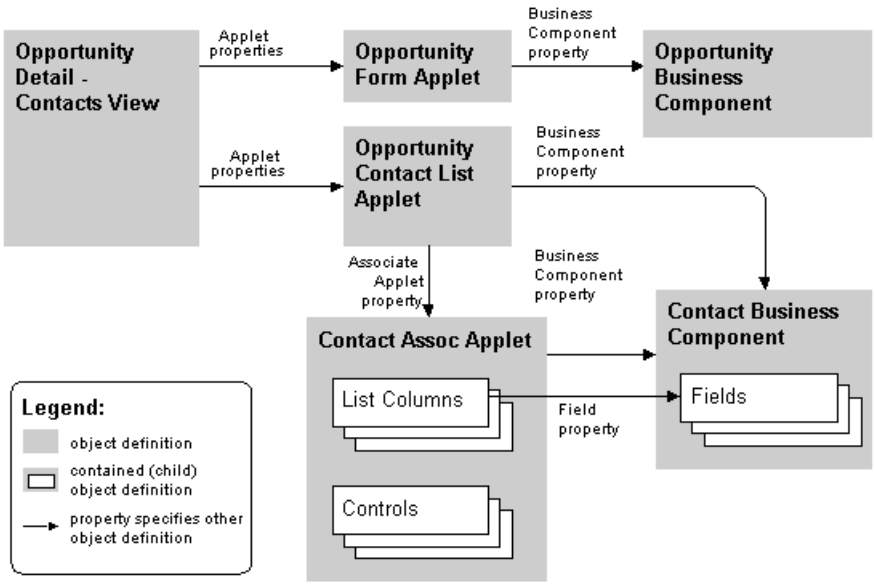


Figure 91. Association Applet (Invoked from Master-Detail View) Details

Roles of the object definitions in Figure 91:

- **View (Opportunity Detail - Contacts List view).** Provides the context in which the association applet is invoked, although no properties of the view directly identify the association applet. The Business Object property of the view establishes the master-detail relationship between the business components whose data is displayed.
- **Master applet (Opportunity form applet).** Form applet that displays one record from the master business component. It has no special properties to configure.
- **Detail applet (Opportunity Contact list applet).** List applet that displays a list of records from the detail business component that are detail records for the current master record in the master business component. The name of the association applet is specified in its Associate Applet property.
- **Business components (Opportunity and Contact).** Provide the data for their respective applets in the view. The detail business component also provides the data displayed in the association applet.

  NOTE: In the association applet, records from the detail business component are displayed; in the detail applet, the only records displayed are those which have already been associated to the current master record.

- **Association Applet (Contact Assoc applet).** Implements the dialog box that appears when the user attempts to add or insert a record in the detail applet. It has a Type property value of Association List, which indicates that it is an association applet, and a Class property value of CSSFrameList, indicating that it is a list applet. The association applet is configured as a standard list applet, with a List child object definition that, in turn, has List Object child object definitions.
- **List columns.** Specify the fields that are displayed in the association applet, and in what order. They duplicate some or all of the list columns in the detail applet in the view.
- **Controls.** Several specialized controls appear in an association applet. They are the following:
  - **Check button.** Associates selected records to the current parent. The result at the table level is to create an intersection table row between the row identified in the master applet and the row identified in the association applet. The control is named PopupQueryAdd and has a method invoked of AddRecord.
  - **New button.** Creates a new row directly in the detail applet. The result at the table level is to create a new row in the detail table, and an intersection table row between the row identified in the master applet and the row created in the association applet. The control is named ButtonNew and has a method invoked of NewRecord.
  - **Cancel button.** Dismisses the dialog box.
  - **Find combo box.** In combination with the Starting With text box and Find button, provides the user with search capabilities for locating the desired record in the association applet. The user selects in this combo box the field to search.
  - **Starting With text box.** Text box where the user enters the search criteria. The criteria entered in this control are automatically completed by a wild card. It is not possible to initiate another search and enter exact search criteria.
  - **Go button.** The user clicks this button to initiate the search specified in the Find combo box and Starting With text box.

Association Applets Invoked from Multi-Value Group Applets in SI Mode

In Standard Interactivity mode, when a user clicks the MVG button and the business component of the underlying MVG applet has a M:M relationship with the master business component, an MVG applet is displayed, as shown in Figure 92.

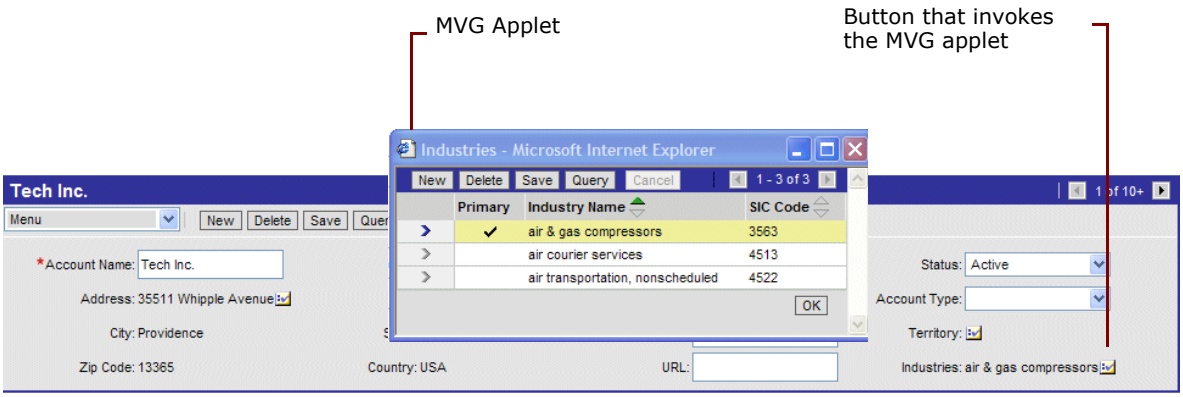


Figure 92. MVG Applet

When the user clicks New in the MVG applet, an association applet appears (shown in Figure 93). The Association applet allows the user to associate a new record to the MVG. Some association applets also allow users to create new records to associate with the MVG.

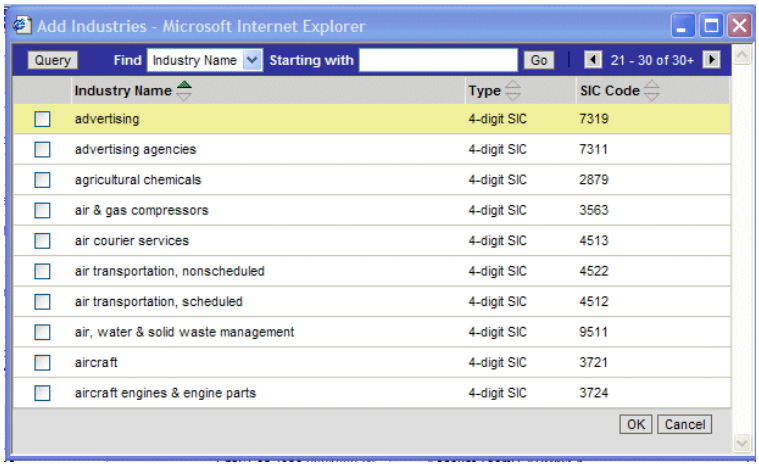


Figure 93. Association Applet (SI Mode)

Figure 94 describes the underlying configuration of an association applet invoked from an MVG applet.

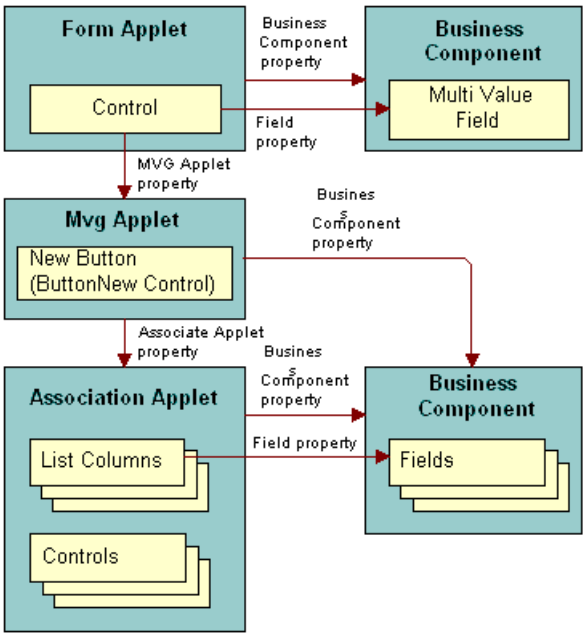


Figure 94. Association Applet (Invoked from Multi-Value Group Applet) Details

The roles of the object definitions in Figure 94 are the following:

- **Form applet.** Contains one or more text box controls displaying multi-value fields. The MVG Applet property for each of these text box controls identifies a multi-value group applet to invoke when the user clicks the button to the right of the text box.
- **Multi-value group applet.** Used to display the list of records assigned to the multi-value field in the form applet. The Associate Applet property in the multi-value group applet's object definition identifies the association applet to invoke.
- **Multi-Value Group business component.** Stores the (detail) multi-value group records for each master business component record. The multi-value group business component supplies records to both the multi-value group applet and the association applet.
- **Association applet.** Used to display list of records available to associate to the parent record. The association applet has a Type property value of Association List, which indicates that it is an association applet. It has a Class property value of CSSFrameList, indicating that it is a list applet. The association applet is configured as a standard list applet, with a List child object definition that has List Object child object definitions.

About Shuttle Applets

In high interactivity mode, when a user clicks the MVG button and the business component of the underlying MVG applet has a many-to-many (M:M) relationship with the master business component, a shuttle applet is displayed (shown in Figure 95). Shuttle applets allow users to associate detail records to a parent record and create new records.

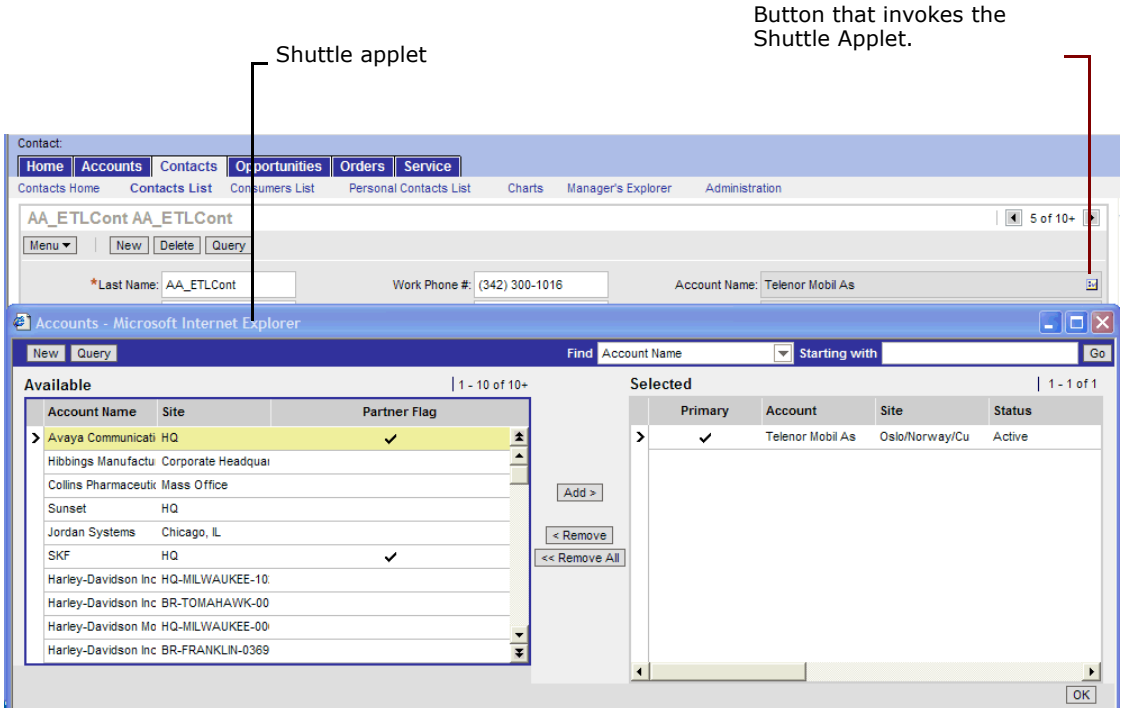


Figure 95. Shuttle Applet

The underlying object architecture for shuttle applets is the same as the architecture described in Figure 94 on page 416 for association applets. However, shuttle applets are rendered using two specialized Web templates (CCPopupListAssoc.swt and CCPopupListMVG.swt). Items on shuttle applets come from the repository object definitions of both the association applet and the MVG applet.

- The following items are taken from the association applet:
  - Applet header (New, Query, Find, Starting With).
  - The *Available* label.
  - The list body on the left side of the shuttle applet.
- The following items are taken from the MVG applet:
  - The Selected label.
  - The list body on the right side of the shuttle applet.

- OK button in the lower right corner.
- The Add, Add All, Remove, and Remove All buttons in the middle.

The Mode property of the Applet Web Template Item object type determines in which applets the controls appear.

- If Mode is not specified, the control appears in both shuttle and non-shuttle applets.
- If Mode is set to DefaultOnly, the control shows in non-shuttle applets only. Examples include the OK and the Cancel button on the association applet.
- If Mode is set to More, the control appears in the shuttle applet only. Examples include buttons such as Add, Add All, Remove, and Remove All.

NOTE: Shuttle applets do not support popup applets. You cannot invoke a popup applet from a shuttle applet.

For additional information related to shuttle applets, see *Release Notes* on Siebel SupportWeb.

20 Configuring Special Purpose Applets

Topics in This Chapter

About Chart Applets

A chart applet graphically displays data from a business component in various formats for analysis of trends, category comparison, and other data relationships. Any data in a business component can be included in a chart. The data in a chart applet reflects the current query for the business component. The user can update the chart with changes to the query by clicking inside the chart. Figure 96 shows a chart applet in a view.

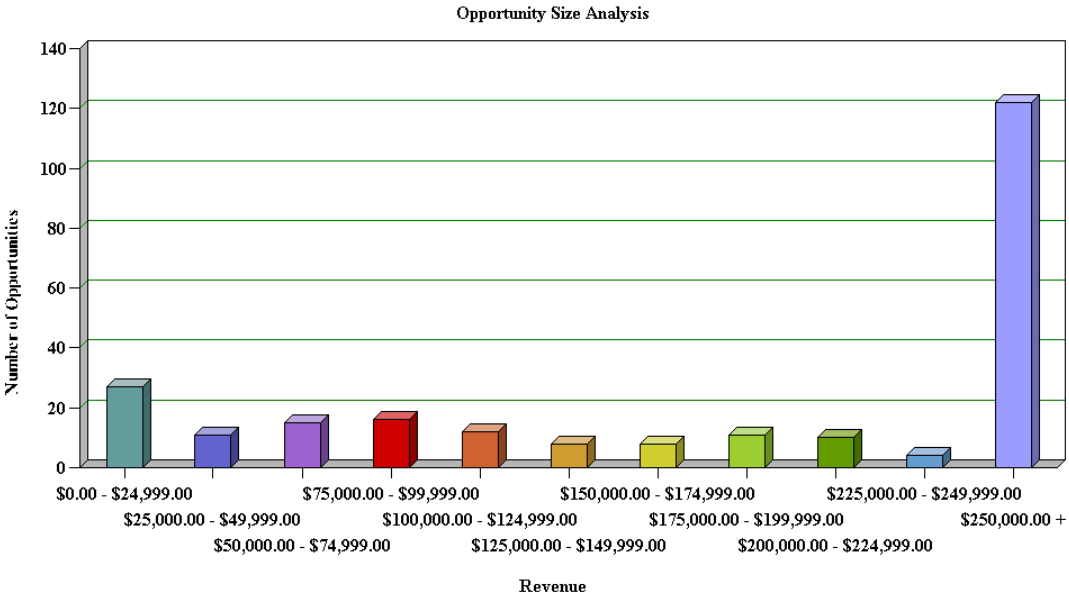


Figure 96. Opportunity Size Analysis View

This view, titled Opportunity Size Analysis (Oppty Chart View - Opportunity Size Analysis in Siebel Tools), lists opportunities in the upper (list) applet and aggregates them by size in the lower (chart) applet. By default, the chart applet in this view (Oppty Chart Applet - Competitor Frequency Analysis) displays the data in bar chart format, in a specific type of bar chart called 3dBar. The user can select different chart types from the Type picklist at upper right in the chart applet. Chart types are discussed in "About Types of Charts" on page 422.

NOTE: To change the size of the legend for a chart applet, right-click on the legend and select one of the options.

Axis Terminology

Specialized terminology is used for axes in Siebel Tools and Siebel applications. Each axis has a special name, as shown in Table 57.

Table 57. Axis Terminology

Axis	Name	Meaning in Bar Charts	Meaning in Line Charts	Meaning in Pie Charts
X axis	Category	The horizontal axis (except in horizontal bar charts, in which the X axis is vertical along the left).	The horizontal axis.	The set of pie slice labels.
Y axis	Data Values	The vertical axis (except in horizontal bar charts, in which the Y axis is horizontal along the bottom).	The vertical axis.	The percentage of the circle occupied by each pie slice, and the corresponding numeric value.
Z axis	Series	A set of labels in the legend. In the stacked bar or cluster bar charts, each series label corresponds to a bar segment or bar of a particular color appearing in each stack or cluster.	A set of labels in the legend. In line charts, each series label in the legend corresponds to one curve.	Do not use a series field with pie charts, because only the first entry in each series will be charted.

An example of a chart with all three axes is the Project Revenue Analysis chart shown in Figure 97.

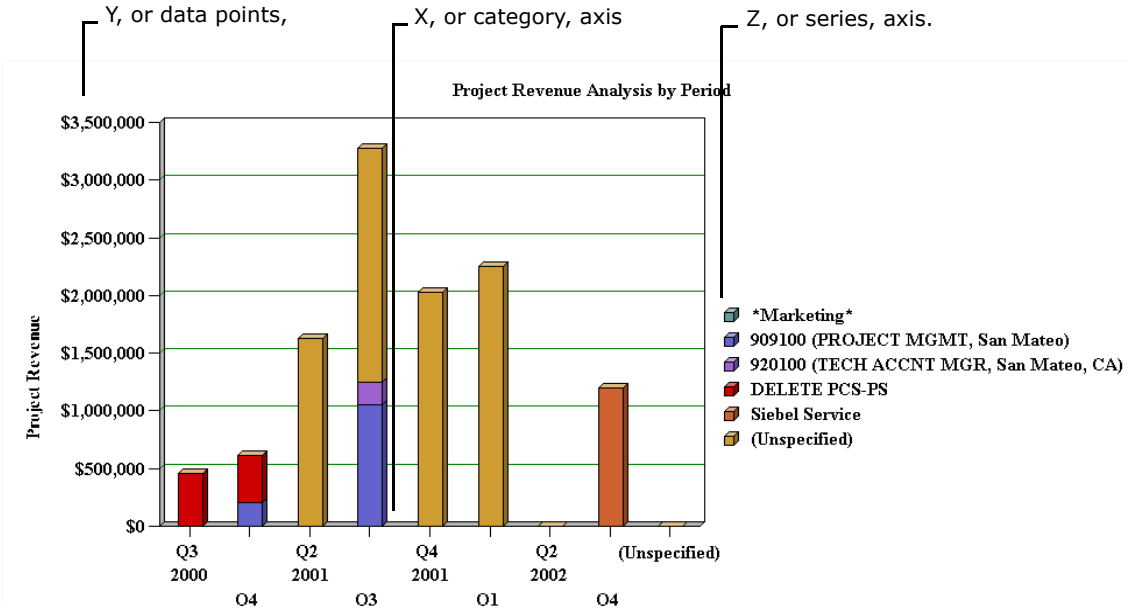


Figure 97. Project Revenue Analysis Chart in Siebel Service

In this chart, the amount of revenue is plotted on the Y (data values) axis, quarters appear on the X (category) axis, and each bar color (Z, or series, axis) identifies a a different project.

NOTE: In charts with two Y axes, the first Y axis refers to the vertical axis on the left side, while the second Y axis refers to the one on the right side.

About Types of Charts

The user can select different chart types from the Type picklist at the upper right in most chart applets. Chart types provide various layout options, including horizontal bar, stacked bar, pie, line, scatter, spline, and combo (combined line and bar). Several of these are available in either 2- or 3-dimensional format. The 3-dimensional types are functionally the same as the corresponding 2-dimensional types, but provide the illusion of bar, line, or pie thickness for visual attractiveness.

The following styles of charts are available (although not all styles are supported for all chart applets).

Bar Charts

Bar charts are typically used to compare the absolute difference in data from one category to another.

- **3dBar.** The 3dBar type divides data from the source records into categories, and displays the total for each category as a vertical bar. This is shown in Figure 98.

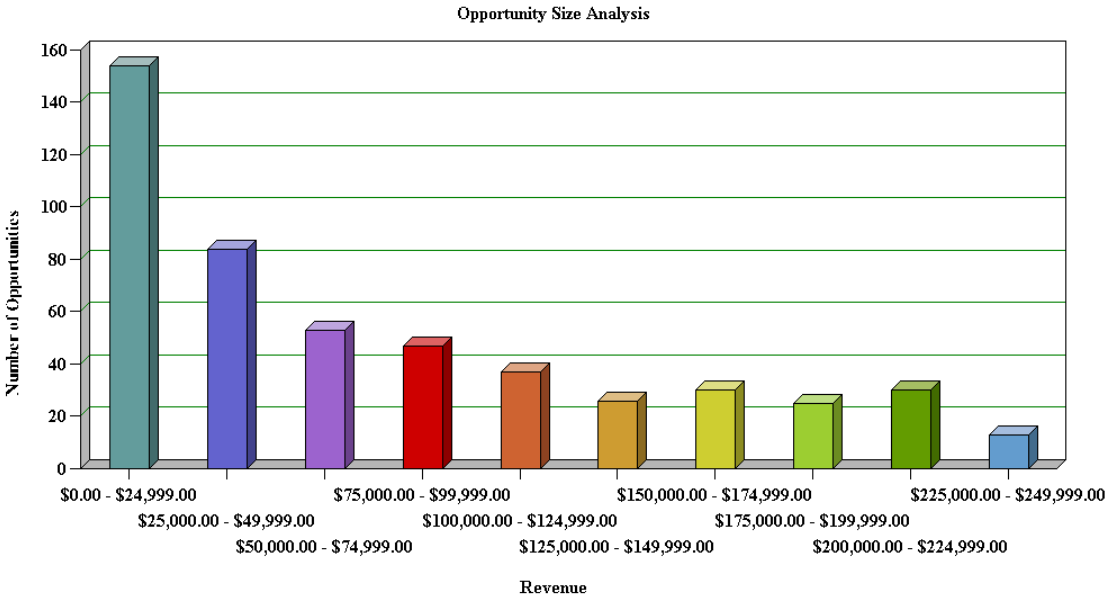


Figure 98. 3dBar Chart

If the chart is configured with a Z (series) axis, a cluster of bars appears for categories rather than a single bar. This is shown in Figure 99.

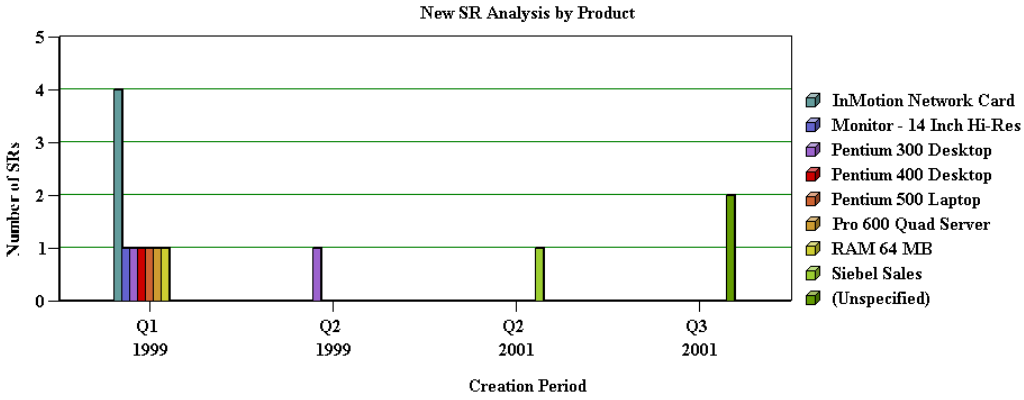


Figure 99. 3dBar Chart with Series Axis

■ **3dHorizBar.** A 3dHorizBar chart is functionally equivalent to a 3dBar chart, but has the X and Y axes switched, with the result that the bars are horizontal. A 3dHorizBar chart appears in Figure 100.

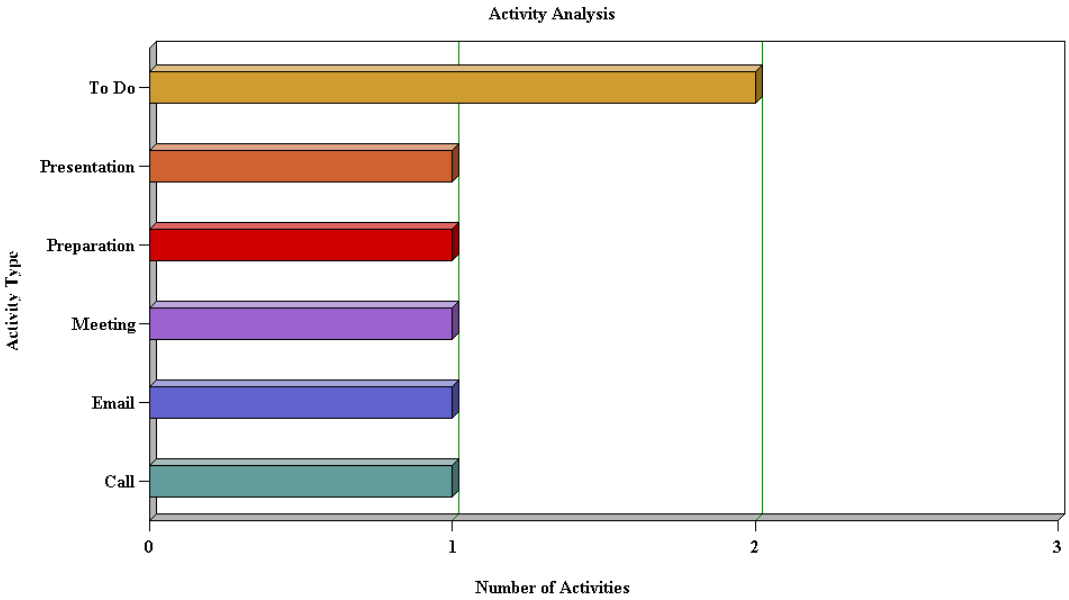


Figure 100. 3dHorizBar Chart

The individual horizontal bars are replaced by clusters of horizontal bars if a series axis is present, as shown in Figure 101.

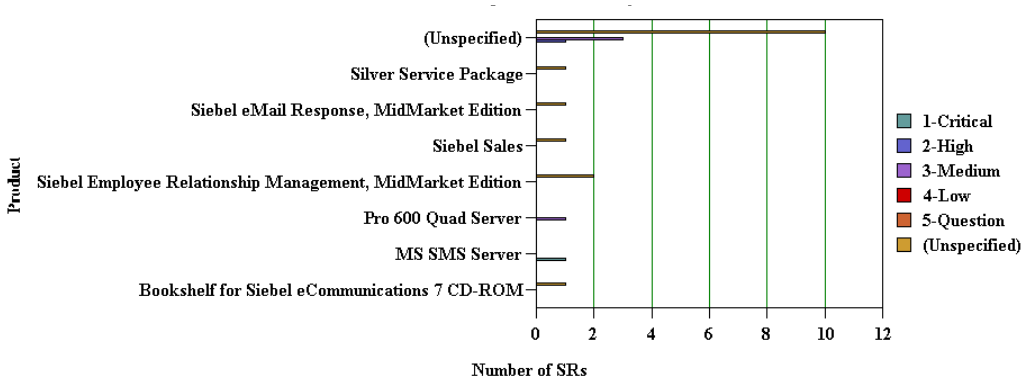


Figure 101. 3dHorizBar Chart with Series Axis

- **3dStackedBar.** A 3dStackedBar chart normally has a series axis. The chart displays a single stack of bars for each category, within which appears a bar of a different color for each series. Stacked bar charts are useful for seeing the individual value for each series within the category as well as their total for the category. An example of a 3dStackedBar chart appears in Figure 102.

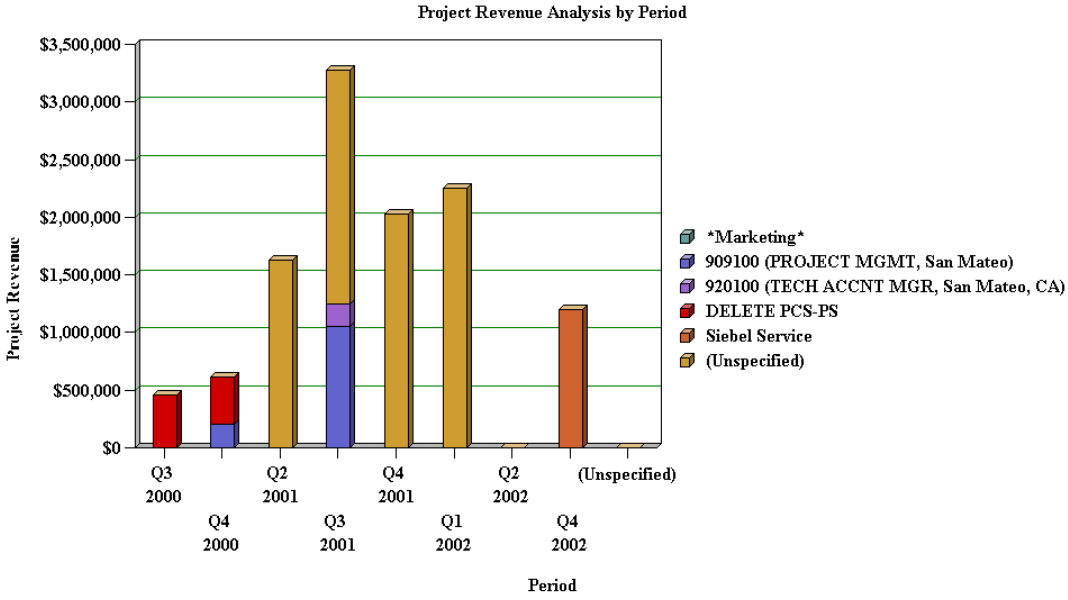


Figure 102. 3dStackedBar Chart

This figure displays a Project Revenue Analysis chart. The data values axis corresponds to project revenue, the category axis corresponds to a quarter, and the series axis corresponds to the project name. So for each quarter along the X axis, there is a stack of bars. Each bar in the stack indicates the revenue reached in a particular quarter. The stacks within each bar indicate the individual projects.

- **2dBar.** A 2dBar chart is functionally equivalent to a 3dBar chart, but is displayed without the illusion of depth. Two-dimensional charts are generally easier to read accurately, but may seem less visually attractive than their three-dimensional counterparts. A 2dBar chart appears in Figure 103.

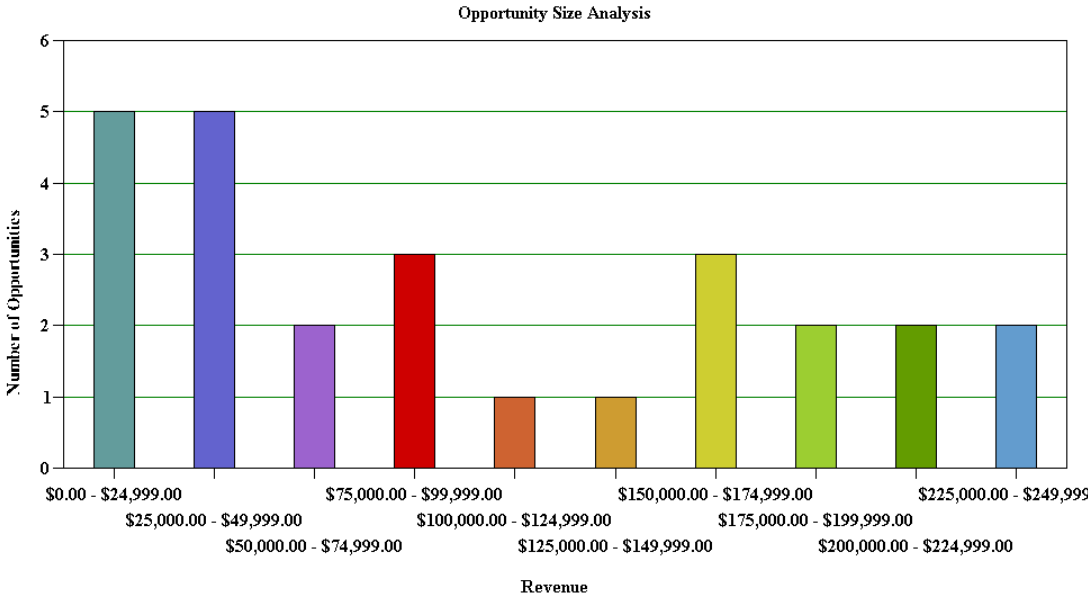


Figure 103. 2dBar Chart

Like the 3dBar chart, a 2dBar chart displays bars in clusters if a series axis is present.

- **2dHorizBar.** The 2dHorizBar chart type is functionally equivalent to the 3dHorizBar type, but is displayed without the illusion of depth. A sample 2dHorizBar chart appears in Figure 104.

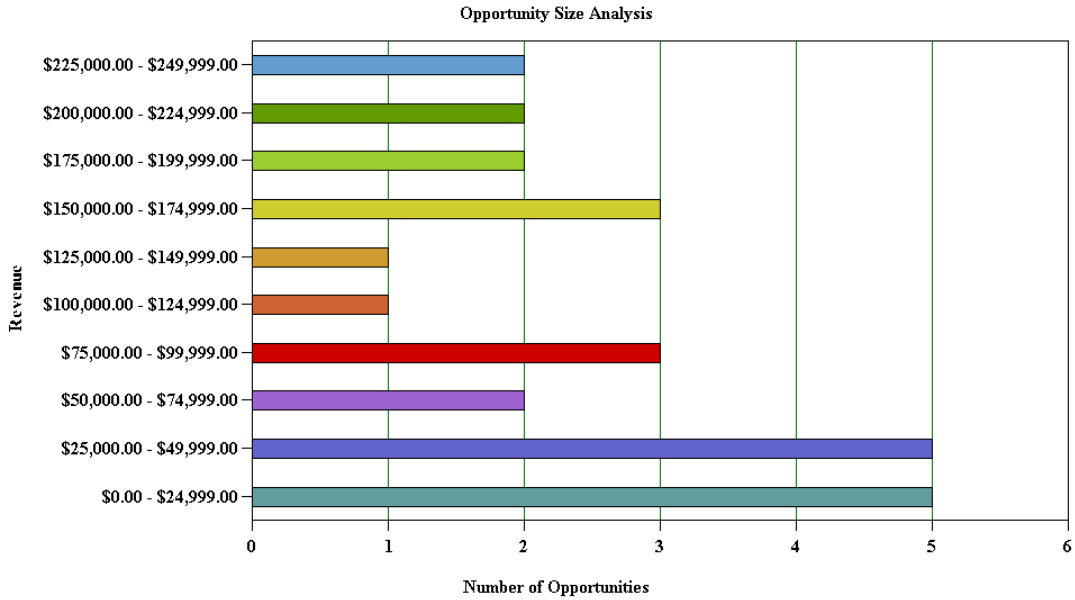


Figure 104. 2dHorizBar Chart

- **2dStackedBar.** The 2dStackedBar chart type is functionally equivalent to the 3dStackedBar type, but is displayed without the illusion of depth. A sample 2dStackedBar chart appears in Figure 105.

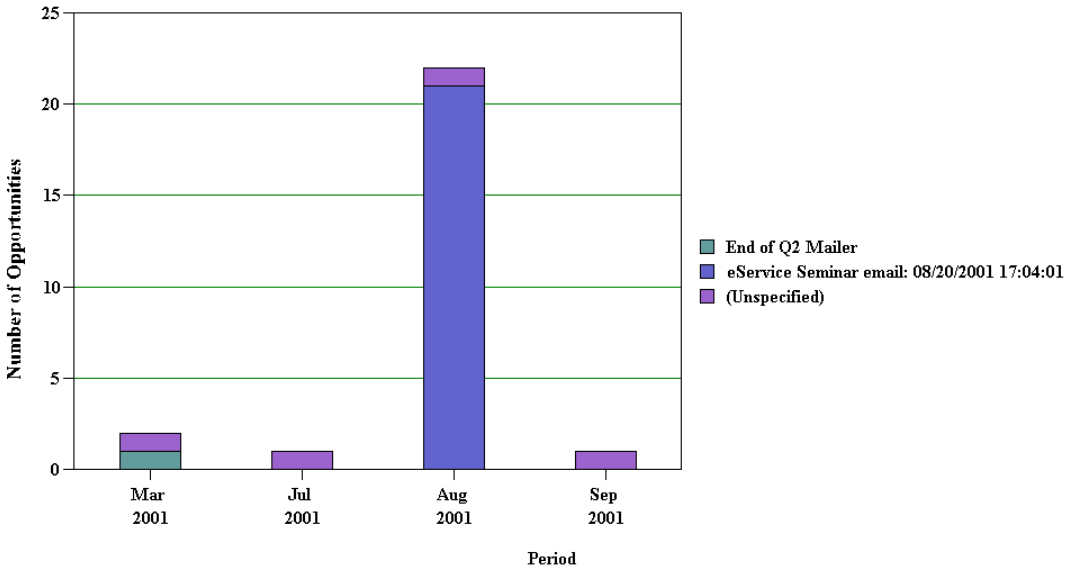


Figure 105. 2dStackedBar Chart

Line Charts

Line Charts are used to observe trends across categories or over time.

- **2dLine.** The 2dLine chart type displays one or more line curves plotted against the X-Y grid. If there is no series axis, a single line curve appears. If there is a series axis, one line curve appears for each color in the legend. A 2dLine chart appears in Figure 106.

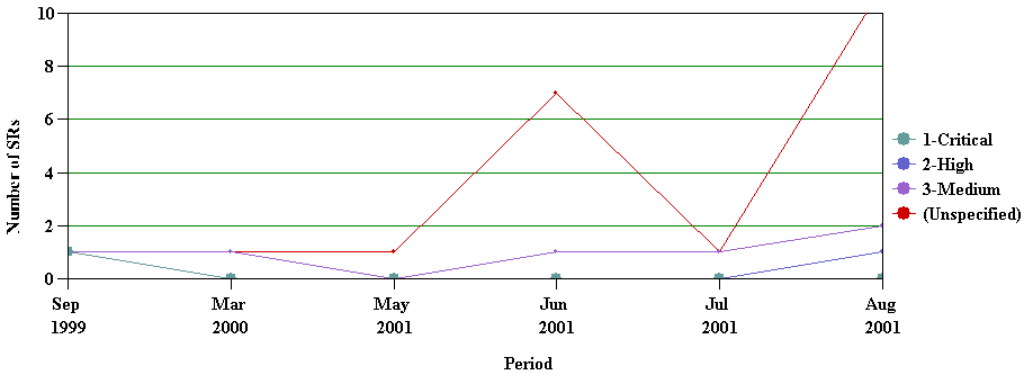


Figure 106. 2dLine Chart

- **3dLine.** The 3dLine chart type is functionally equivalent to the 2dLine type, but appears with the illusion of depth. A 3dLine chart (showing the same data as the 2dLine chart in Figure 106) appears in Figure 107.

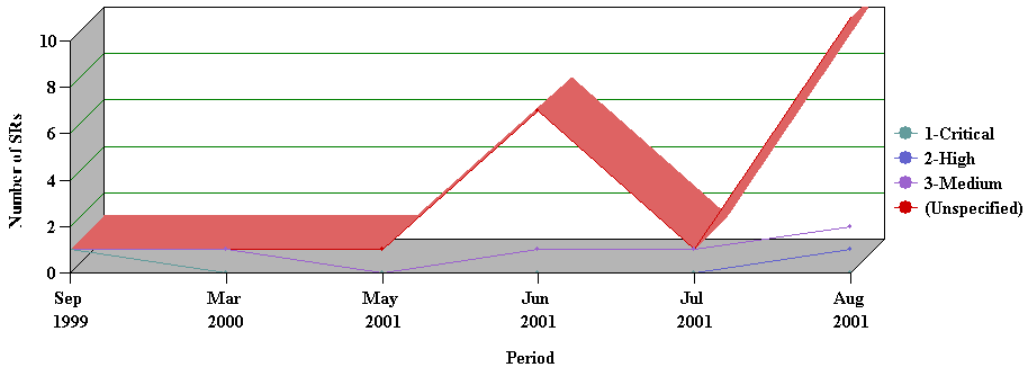


Figure 107. 3dLine Chart

- **2dSpline.** The 2dSpline chart type displays one or more line curves plotted against the X-Y grid, with the points plotted accurately but the line between them smoothed mathematically. If there is no series axis, a single curve and set of points appear. If there is a series axis, one curve and corresponding set of points appear for each color in the legend. A 2dSpline chart appears in Figure 108.

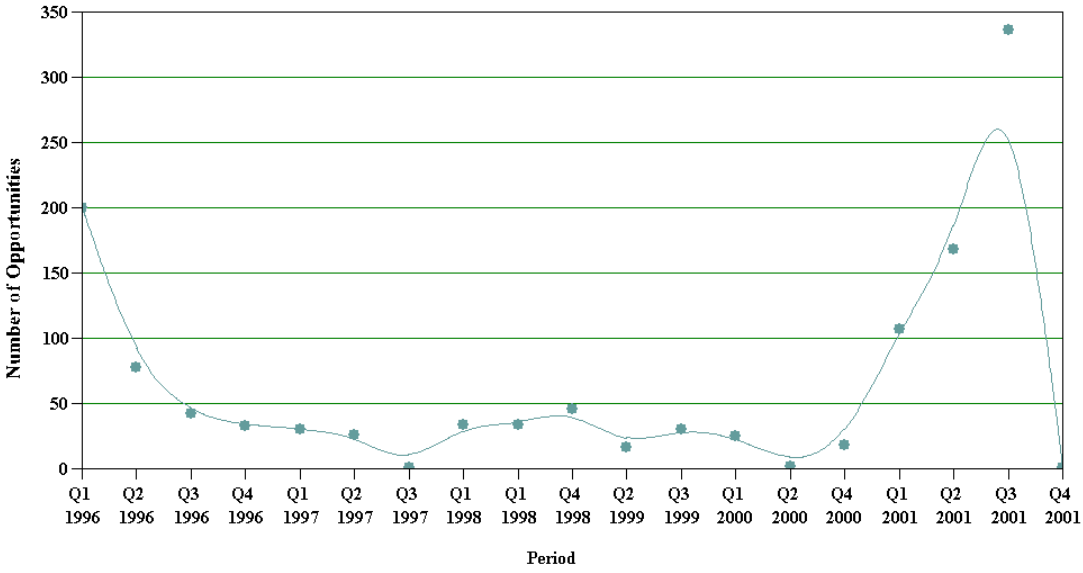


Figure 108. 2dSpline Chart

- **3dSpline.** The 3dSpline chart type is functionally equivalent to the 2dSpline type, but appears with the illusion of depth, and does not display the actual data points, only the smoothed curve. A 3dSpline chart (showing the same data as the 2dSpline chart in Figure 108) appears in Figure 109.

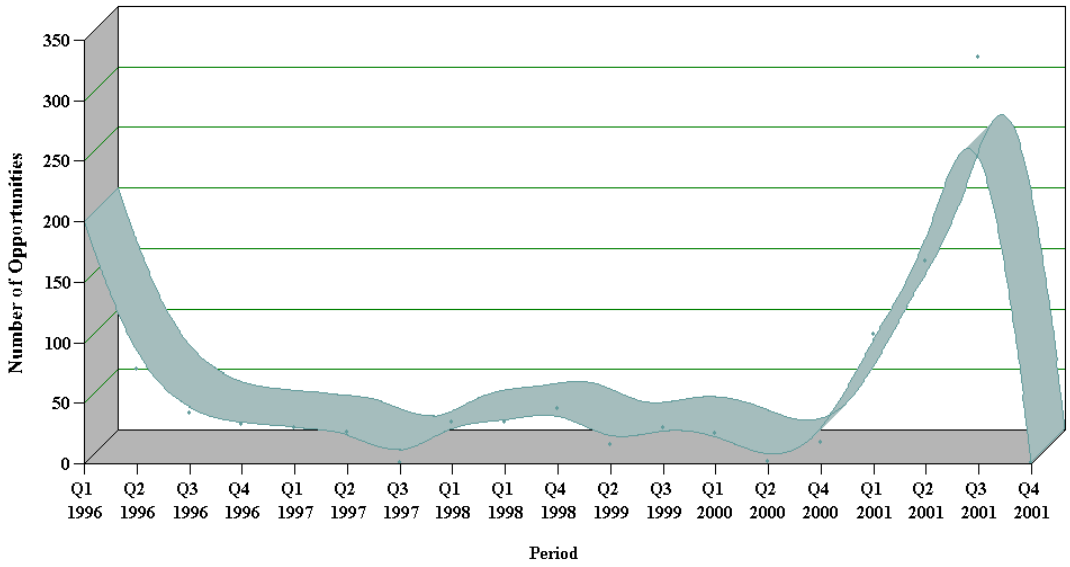


Figure 109. 3dSpline Chart

- **Combo.** A chart of the Combo type displays a single bar chart with dots superimposed on it. The two charts share the category axis, but each has its own data points axis (on the left for the bar chart, and on the right for the line chart). A sample Combo chart appears in Figure 110.

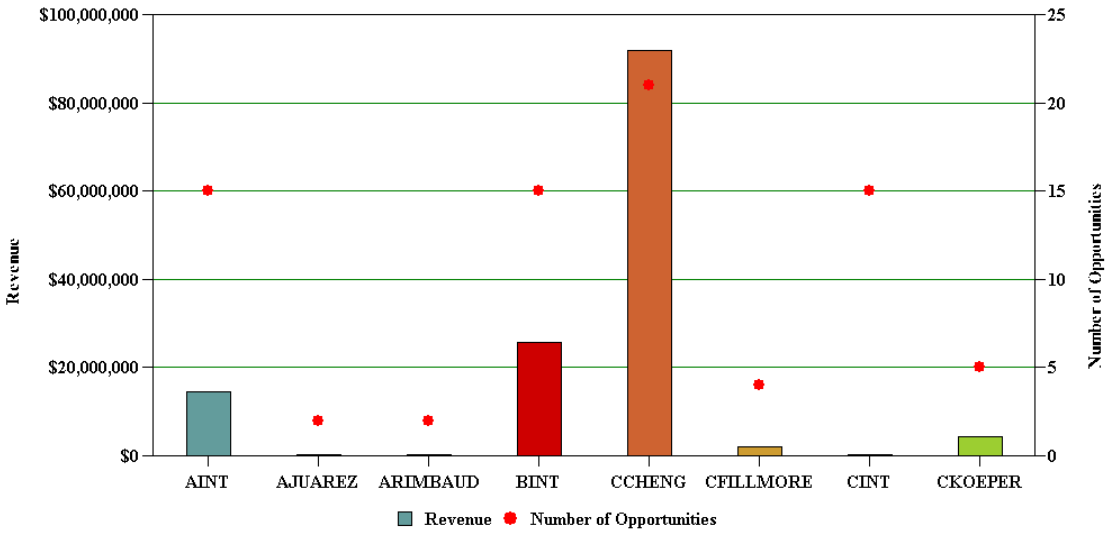


Figure 110. Combo Chart

Pie Charts

Pie Charts are used to compare the relative difference across categories by dividing a circle into segments that represent each category's percentage of the whole.

- **3dPie.** The 3dPie chart type aggregates data point data in the records by category, and displays each category as a separate segment in the pie. The category (X) axis is the set of pie slices and corresponding labels. The data points (Y) axis determines the relative size of each pie slice as a percentage of the total. You cannot specify a series axis for pie charts. The 3dPie chart type gives the illusion of depth, for visual attractiveness. A sample 3dPie chart appears in Figure 111.

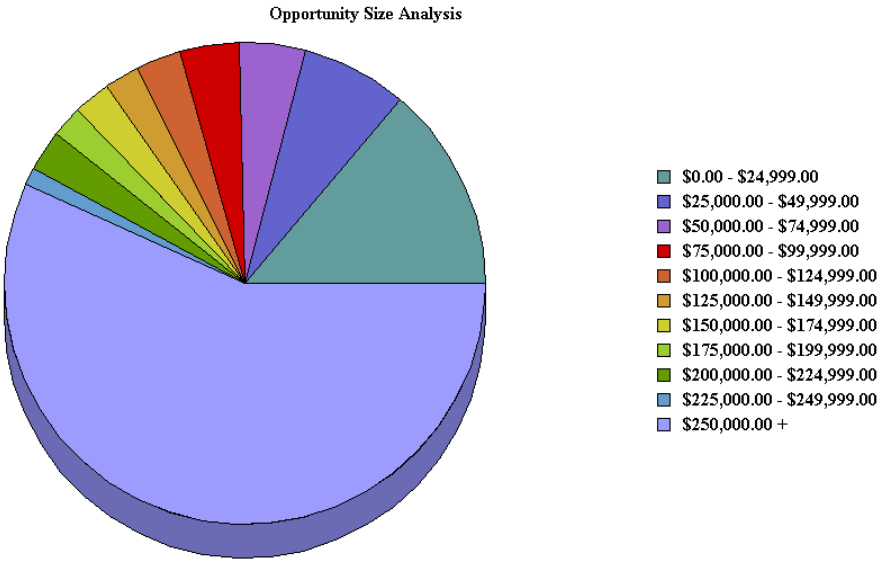


Figure 111. 3dPie Chart

- **2dPie.** The 2dPie chart type is functionally the same as the 3dPie type, but without the illusion of depth. A sample 2dPie chart appears in Figure 112.

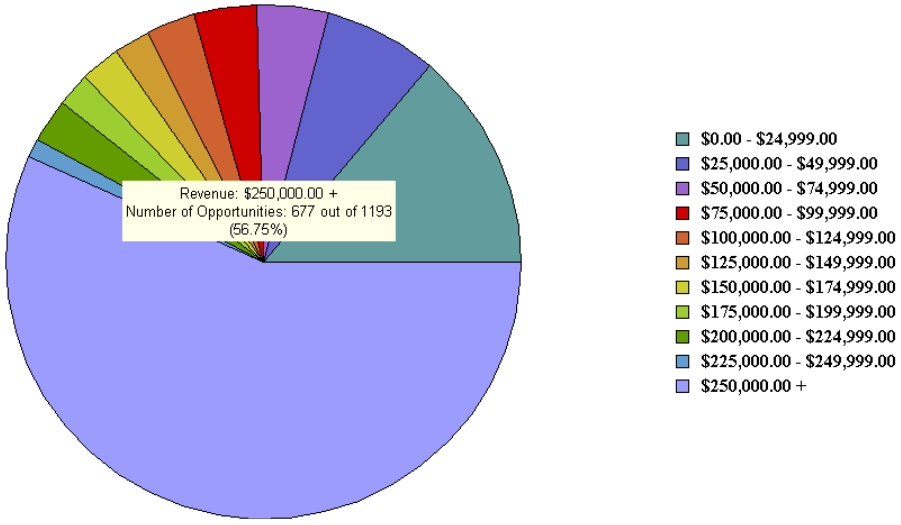


Figure 112. 2dPie Chart

Scatter Charts

- **2dScatter.** A scatter chart—a chart with the 2dScatter type—displays the distribution of data according to two attributes. This is useful for probability distributions, among other applications. The category axis must contain numeric data, as opposed to date or text data. This makes the 2dScatter type unsuitable for conversion to other chart types such as bar, line, or pie. For this reason, the 2dScatter type does not appear in Type picklists, and a 2dScatter chart does not have a Type picklist. A 2dScatter chart appears in Figure 113.

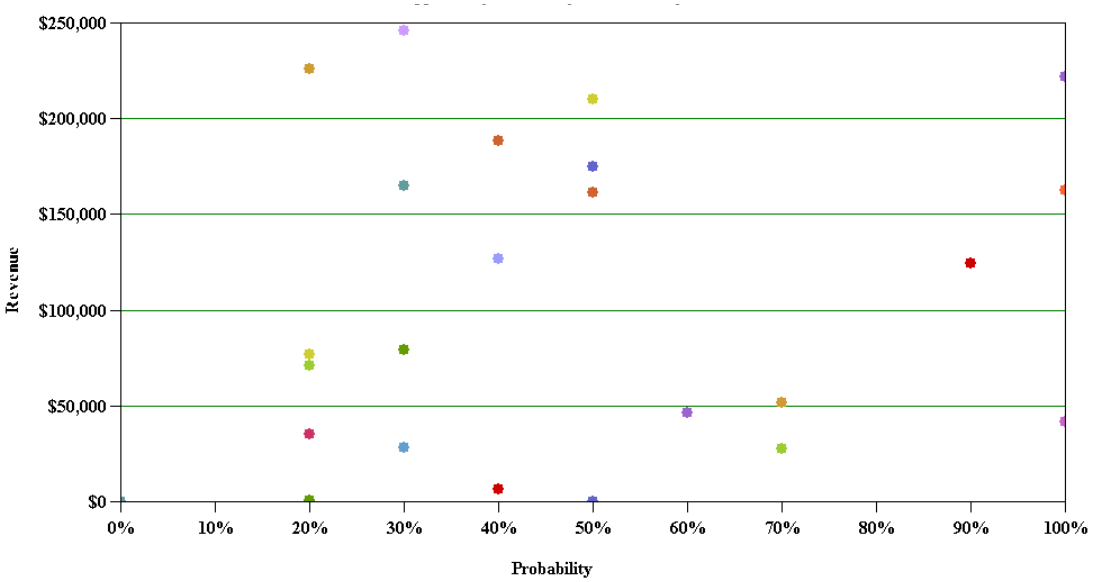


Figure 113. 2dScatter Chart

Understanding How Chart Applets Are Constructed

A chart is built as an applet containing one or more Chart object definitions. The Chart object type is a child of applet. The Chart object type has Chart Element children. This section describes how chart applets are configured.

Business Component Mapping

A chart applet has, like all applets, a business component identified in its Business Component property. Records in this business component—subject to the current view, the current query, and visibility considerations—provide the data displayed in the applet. In the case of a chart applet, specific fields are used to provide the data for the category, data point, and series axes. The correspondence between axes and fields is specified in properties in the Chart object definition.

In the simplest case—a single bar or line graph, with no series axis—a category field and a data point field are specified. Pairs of category and data point field values are plotted as points or bars. If multiple records have the same category value, their data point values are added together.

The Oppty Chart Applet - Source Analysis applet provides an illustration of this process (Figure 114).

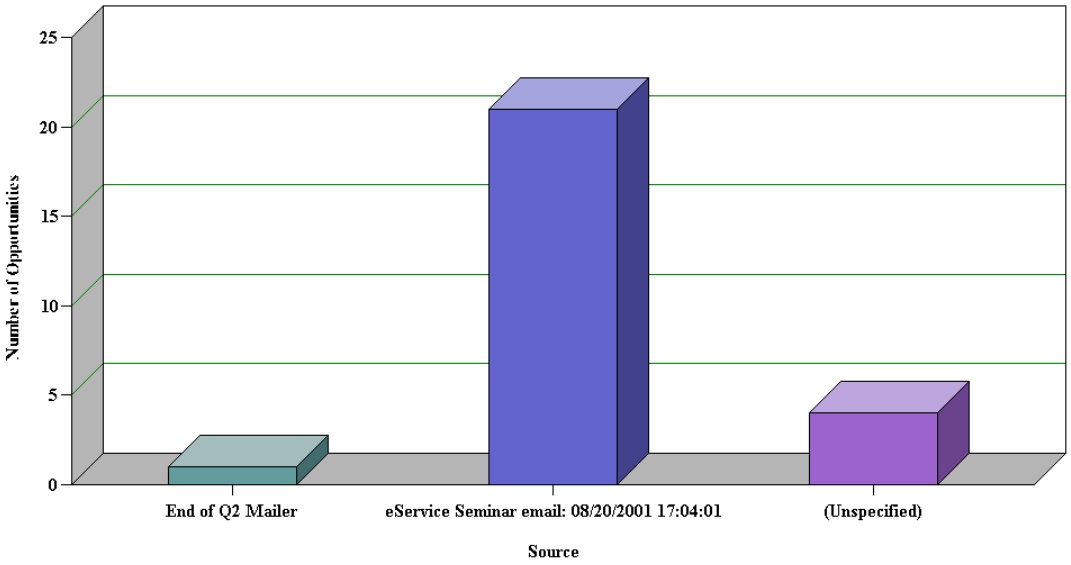


Figure 114. Oppty Chart Applet - Source Analysis

This applet displays the number of opportunities on the data point axis plotted against the source of the opportunity (referral, magazine article, Web site, and so on) on the category axis. To generate the data required for the curve, the Source field in each record is checked and the number of opportunities for each distinct source value is tallied. The result is a two-row temporary table with a column for each source, as shown in Figure 115.

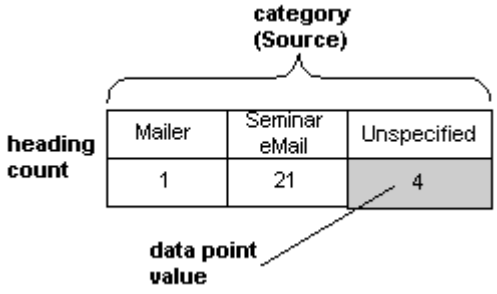


Figure 115. Temporary Table for Single-Curve Chart Data

For a multiple-curve chart, a row is added to the temporary table for each curve in the series (Figure 116).

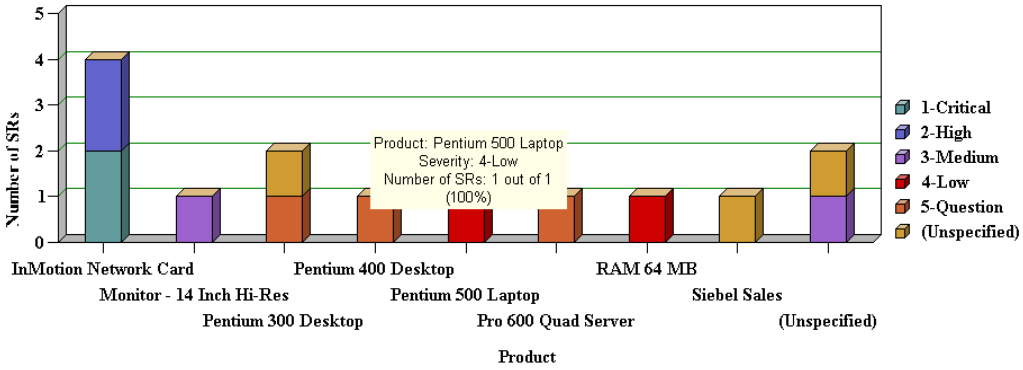


Figure 116. Multiple-Curve Chart

The temporary table for a multiple-curve chart is illustrated in Figure 117.

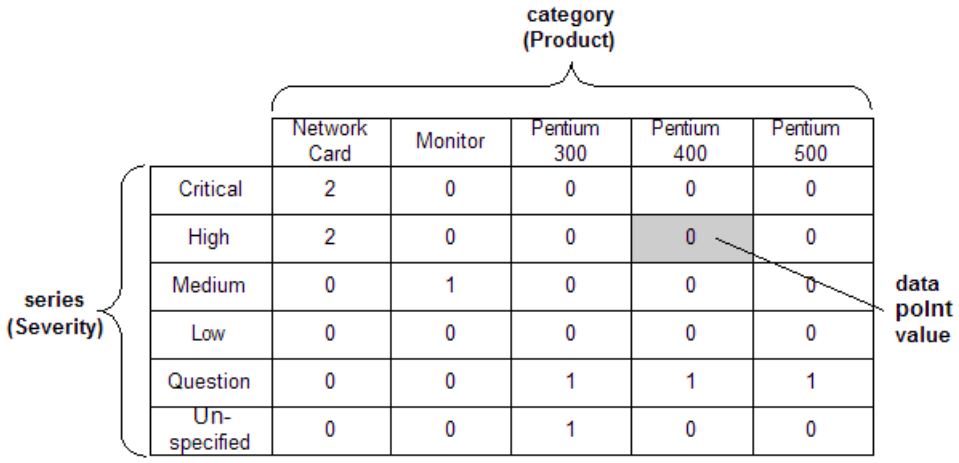


Figure 117. Temporary Table for Multiple-Curve Chart Data

To define the data mapping from the business component into the chart applet, you need to define the following properties in the Chart object:

- **Category Field.** Contains the name of a text or date field in the business component (except for scatter charts, which use a numeric category field). When the business component records are scanned, the different values found in this field are mapped into different categories. These values are displayed on the chart's X-axis labels.

- **Data Point Field.** Contains the name of a numeric field in the business component, or is unspecified. If specified, the value in this field in each record is added to the total for the category field value in the same record. If a data point field is not specified, the count for the corresponding category field is incremented rather than adding the data point value to the total for the category field. These counts or totals determine the height along the Y-axis of a bar or line curve point for each unique category field value in the curve. Rather than a total or a count, some other function (specified in the Data Function property) may determine the use of the data point field data.
- **Series Field.** Contains the name of a text field in the business component, or is unspecified. When the business component records are scanned, the different values found in this field are mapped into different curves. These values are displayed on the chart's legend labels.

  NOTE: The maximum number of Series cannot exceed 50 when running the chart. If it does exceed 50, an error message is displayed. The user may have to run another query that results in less than 50 Series.

- **Data Function.** The Data Function property determines how the data point field values are converted into the new table's cell values. Possible values are Sum (simple addition), Count (number of occurrences of a cell value), Average (average value per record), and Plot (different from Count only in that when a cell is empty, it is charted as NULL instead of 0).

The preceding descriptions cover the use of these properties for the most general cases. There are a number of special cases in which these properties are configured differently than described. Some special case configuration scenarios are described in the sections that follow. For descriptions of the properties, see *Object Types Reference*.

Using Picklists in Chart Applets

A chart applet typically provides one or more picklists along the upper edge that allow the user to reconfigure the chart's presentation or use of data. These picklists are illustrated in Figure 118.

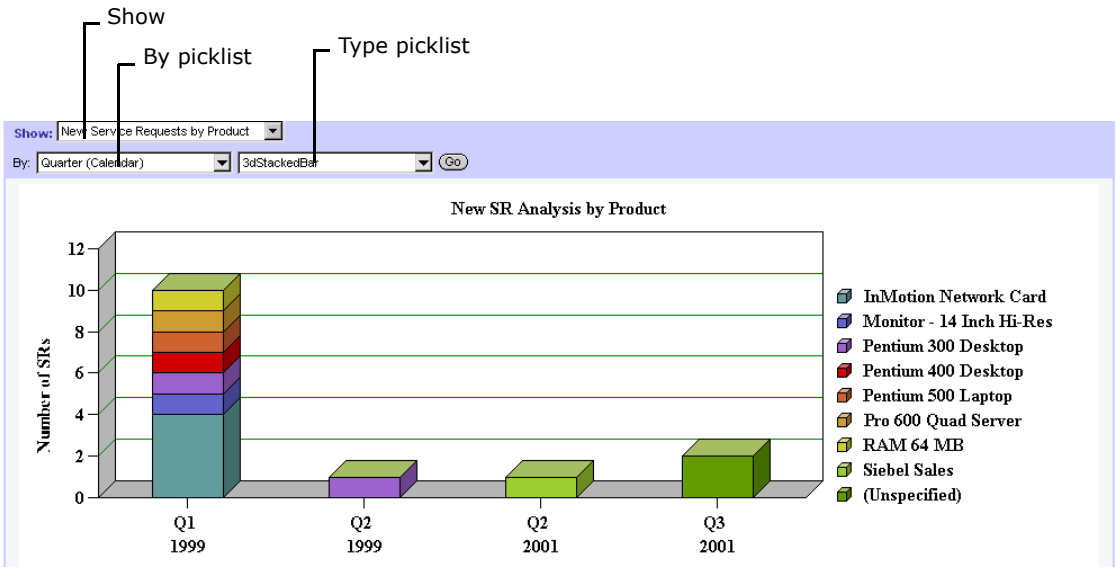


Figure 118. Picklists in a Chart Applet

These picklists are described as follows:

- **Type picklists.** This is the most common of the four picklists, and appears in most chart applets. It provides the user with the means to select a different type of chart for the same data, such as a pie chart instead of a bar chart, or a two-dimensional line chart instead of a three-dimensional one. The chart types are described in detail in "About Types of Charts" on page 422.

  The options for the Type picklist are specified in the Picklist Types property of the Chart object definition, as a comma-separated list of chart type names such as the following:

  ```
  3dBar,3dStackedBar,3dPie,3dHorizBar,2dBar,2dStackedBar,2dPie,
  2dHorizBar
  ```

  There cannot be any spaces between the elements in the comma-separated list.

  The default type—the chart type to appear when the chart is initially displayed—is specified in the Type property. Charts without a Type picklist use the Type property to specify the chart type of the chart; in that situation the chart type cannot be changed by the user.

- **Show picklists.** This picklist allows the user to change what is displayed on the Y axis. The choices available depend on the configuration of certain properties in the Chart object definition. The Show picklist displays a selection list of field and function combinations which determines what values are plotted along the Y axis.

  For information on configuring the Show picklist, refer to "About Show Picklists" on page 438.

- **By Picklist.** This picklist allows the user to change what is displayed on the X axis. This can provide any one of three roles, depending on the configuration of certain properties in the Chart object definition:
  - In period Charts, the By picklist is populated with different periods. This allows the user to select from a list of possible X-axis periods for calendar (day/week/month/quarter/year) data. This requires selection options to be specified in the Picklist Periods property in the Chart object definition.
  - When a list of source fields is specified rather than a single source field, the picklist allows the user to choose which source field populates the X axis.
  - It can allow the user to invert the X and Z axes, so the user can see the data from a source field in a business component displayed along the X or Z axis per the picklist selection.

  For information on configuring the By picklist, refer to "About the By Picklist" on page 440.
- **Second "By" picklists.** This picklist allows the user to choose which source field populates the Z axis. For information on configuring the second By picklist, refer to "About the Second By Picklist" on page 441.

Each of the four picklists requires a corresponding control of type ComboBox, as a child object definition of the chart applet. Each has required values in the Name and MethodInvoked properties, as detailed in Table 58.

Table 58. Name and MethodInvoked Properties for Four ComboBox Controls

Picklist	Control Name	MethodInvoked
Type	ChartPicktype	PickChartType
Show	ChartPickfunction	PickYAxis
By	ChartPickby	PickXAxis
By #2	ChartPickby2	PickZAxis

About Show Picklists

The Show picklist (the combo box control named ChartPickfunction) can be configured to display a selection list of field and function combinations, the selection from which determines what values are plotted along the Y axis. Multiple combinations of source field and function are provided in the selection list. The Y axis title is obtained from the text in the user's Show picklist selection.

To configure the Show picklist, the following three properties of the Chart object definition are used:

- **Data Point Field.** You enter a comma-separated list of source fields, one for each entry that is to appear in the Show picklist. The first entry in the list is the default. If only one field name is entered, it applies to all functions in the picklist.

- **Data Function.** You enter a comma-separated list consisting of the following function names: SUM, COUNT, AVERAGE, or PLOT. PLOT indicates that the Y values are derived directly from the values in the source field. The order in the comma list determines the association with a data point field and title (picklist function). If the comma-separated list is omitted or it contains fewer elements than the list of names in the Picklist Functions property, the list `Sum,Count,Average,Plot` is substituted.
- **Picklist Functions.** You enter a comma-separated list of Y-axis titles, which are also the text which appears in the picklist. The order in the comma list determines the association with a data point field and data function.

For example, you could configure a Show picklist with explicit syntax that offers three choices: Number of Opportunities, Opportunity Revenue, and Opportunity Expected Revenue. This is configured with the property settings shown in Table 59.

Table 59. Show Picklist Properties for Sales Method Bar Chart

Property	Value
Picklist Functions	Number of Opportunities, Opportunity Revenue, Opportunity Expected Revenue
Data Function	Count,Sum,Sum
Data Point Field	Name,Revenue,Expected Revenue

As can be seen from the table, there are three values in each comma-separated list. The first entry, Number of Opportunities, performs a Count function on the Name field. The second entry, Opportunity Revenue, performs a Sum function on the Revenue field. The third entry, Opportunity Expected Revenue, performs a Sum function on the Expected Revenue field.

An example of a Show picklist configured with implicit syntax and the standard function list is in the Lead Source Analysis chart in the Opportunity New Business Analysis view in Siebel Sales (Oppty Chart Applet - New Business). The picklist offers three choices: Number of Opportunities, Opportunity Revenue, and Average Opportunity Revenue. This is configured with the property settings shown in Table 60.

Table 60. Show Picklist Properties for Lead Source Analysis Chart

Property	Value
Picklist Functions	Number of Opportunities,Opportunity Revenue, Avg Opportunity Revenue
Data Function	Count
Data Point Field	Revenue

The value of Revenue in the Data Point Field property applies to all entries in the picklist.

The value of Count in the Data Function property is unnecessary; it could be left blank instead. Whenever the number of entries in the Data Function property is not the same as the number in the Picklist Functions property, the system supplies a standard Data Function list. This list is the following:

```
Count,Sum,Average,Plot
```

The first picklist entry, Number of Opportunities, performs a Count function on the Revenue field. The second entry, Opportunity Revenue, performs a Sum function on the same field. The third entry, Avg Opportunity Revenue, performs an Average function.

This means of configuring Show picklist behavior predates the ability to specify triplets of name, function, and field, and is more restrictive. It has been retained for backwards compatibility with earlier versions of Siebel applications. Generally it makes more sense to explicitly specify the values in the three properties.

About the By Picklist

The contents of the Category Field property in the Chart object definition determine the behavior of the By picklist (ChartPickBy combo box control), as follows:

- **Calendar increments in the picklist and X axis**. If the Category Field property contains the name of a single field that has a DTYPE_DATE data type, the X axis displays calendar increments and the chart is considered a *period chart*. In this situation, the picklist is populated with calendar increment options, including user defined periods (specified in View > Site Map > Application Administration >Periods) such as Day, Week, Month, Quarter, and Year.

  For example, in the New Business Analysis chart, the category field is Created (the date of creation of the record, hence of the opportunity). As a result, the category axis contains date increments, based on the increment the user selects in the By picklist.

- **Text labels in the X axis, category and series field names in the picklist**. If the Category Field property contains the name of a single text field from the business component, and a series field has also been specified (in the Series Field property), the By picklist is populated with the names of the category field and the series field. The user can select either field to populate the X axis with labels derived from the contents of that field; the unselected field populates the legend box (Z axis) with labels. The category field is the default, and is initially displayed on the X axis.

  For example, the chart in the Service Request Product Analysis view in Siebel Service has a category field of Product and a series field of Severity. When the chart is initially displayed, the X axis labels are product names and the legend labels are severity levels. However, the field names Product and Severity appear in the By picklist, and the latter selection allows the user to display severity levels in the X axis and product names in the legend.

- **Text Labels in the X axis, multiple field names in the picklist**. If the Category Field property contains a comma-separated list of field names, the user is provided with this list of fields at run time in the By picklist. The user's selection determines the field which populates the X axis. The first value in the comma-separated list is the default. (You should avoid blank spaces before or after field names in the list.)

- **Numeric values in the X axis, no picklist**. If the Category Field property contains the name of a single numeric field, the X axis is populated with numeric increments, similar to the process of generating increments for the Y axis. In this situation, the By picklist is not shown.

  For example, the Probability Cluster Analysis chart in the Opportunity Probability Cluster Analysis view has a category field of Rep % (the probability of a sale). In this chart, probability is plotted against the X axis, the X axis increments are percentages from 0% to 100%, and no By picklist appears.

About the Second By Picklist

The contents of the Series Field property in the Chart object definition determine the behavior of the second By picklist (the combo box control named ChartPickBy2), as follows:

- If the Series Field property is blank, all records are mapped into a single series.
- If the Series Field property contains the name of a field from a business component, the Z axis (legend) is populated with labels derived from the contents of that field.
- If the Series Field property contains a comma-separated list of field names, the user is provided with this list of fields at run time in the second By picklist. The user's selection determines the field which populates the Z axis. The first value in the comma-separated list is the default.

Charts with Multiple Curves Plotted Against One Y Axis

Multiple line graph curves can be plotted against the same Y axis, based on different source field/function combinations. The name for each curve appears in the legend. For example, you may want revenue, expected revenue, and net profit to appear as superimposed curves on the same line graph. To accomplish this, set the following property values in the Chart object definition:

- **Data Point Field.** Provide a comma-separated list of source fields, one for each curve to appear in the graph.
- **Data Function.** Provide a comma-separated list consisting of some of the following function names: SUM, COUNT, AVERAGE, or PLOT. PLOT indicates that the Y values are derived directly from the values in the source field. The list of function names must have the same number of entries as the Data Point Field list. The order in the comma list determines the association with a data point field and title.
- **Picklist Functions.** Provide a comma-separated list of Y-axis titles, which identify the individual curves in the Legend. The list of titles must have the same number of entries as the Data Point Field list. The order in the comma list determines the association with a data point field and data function.
- **Series Field.** Remove any existing value(s) from this property; it must be blank. Otherwise, the multiple curves are converted to a Z axis.
- **Multi Data Point.** Set to TRUE. This indicates that multiple curves are to be plotted.

You should also remove the Show combo box and its label in the Applet Web Editor.

Charts with Two Y Axes

Two line graph curves can appear in the same Chart, plotted against different Y axes (one to the left of the graph, the other to the right). Any field or function combination can be used on the left Y axis, and likewise for the right. To accomplish this, set the following property values in the Chart object definition:

- **Data Point Field.** Specify two fields, separated by a comma. The first is for the left Y axis, the second is for the right Y axis.
- **Data Function.** Specify two functions, separated by a comma. The first is for the left Y axis, the second is for the right Y axis.
- **Type.** Set to Combo.

Axis Points—Limiting and Sorting

The number of X axis (category) or Z axis (series) labels can be limited to some predefined number. This can be useful if you are interested in displaying only the *N* highest or *N* lowest values for some field or calculated Y value. For example, you could display the 10 highest revenue accounts by charting the Revenue field in descending order and limiting the X axis to 10 data points. This is accomplished using two properties of the axis label Chart Element for the appropriate axes, as follows:

- **Divisions property (X or Z axis).** Enter an integer to limit the number of X axis or Z axis labels to the number you enter. Note that the AxisId property must be either XAxis or ZAxis, and the Type property must be AxisLabel.
- **SortSpecification property (Y axis).** Enter a value of Ascending or Descending. Note that the AxisId property must be set to YAxis and the Type property must be AxisLabel.

You can set up a sort specification on the Y axis independent of limiting the number of X or Z axis divisions. A sort specification on Y will order the data points regardless of whether you are limiting the display to the first *N* points. The converse is not true, however; it would not make sense to set a number of X or Z axis divisions without also setting a sort specification on Y.

You also can sort on X axis or Z axis labels instead of Y axis values. To accomplish this, you set the Sort Specification in the X axis (or Z axis) label Chart Element object definition rather than in the Y axis label. For example, if the X axis is displaying country names, they can appear alphabetically from left to right. This is different from sorting on Y axis values, which are numeric values from a field in a business component or function based on that field.

Chart Element Object Type

Chart Element is a child object type of Chart. The following types of Chart Elements (as specified in the Type field in the Chart Element object type) are supported:

- **AxisLabel.** Displayed along each axis, with one label for each division of the axis.

- **AxisLineGrid.** Grids make it easier to comprehend a Chart. You can set various grid properties, such as grid color, width, and visibility, on an axis-by-axis basis.
- **AxisTitle.** Displayed along each axis, with one title per axis.
- **Graphic.** A line, rectangle, or ellipse used to emphasize a region of the Chart.
- **Legend.** The list of colored rectangles with accompanying category labels on the left side of the Chart.
- **Plot.** The area that contains the graphs, usually in the center of the Chart.
- **Title.** The large string of text, usually at the top of a Chart.
- **Font, Color, and Size.** For most Chart Elements that contain text, you can set such text properties as font, color, and size.
- **Fill color.** You can set the fill color of the Chart and Plot Chart Element types.

The properties of the Chart Element that apply to the axis label for the X axis (Coordinates, Display Format, Divisions, List Of Values, Sort Specification, and Text) should not be used when specifying a list of X axis source fields, as they can be relevant only for one X axis field. Also, the text of the X axis title is determined dynamically from the combo box selection if the By combo box provides a list of source fields. Whatever is in the Text property in the AxisTitle chart element for the X axis is overridden at run time.

The same restrictions are relevant for the Z axis.

Making X-Axis Labels Vertical

You can make x-axis labels vertical so that they do not overlap with each other. To do this, set the Vertical property to TRUE for the Chart Element object whose Axis Id property is set to XAxis.

Sizing Chart Images

You can change the size of a chart applet by setting the HTML Width and HTML Height properties (in pixels) for the Chart control child object of the applet.

The default values are 1012 for HTML Width and 560 for HTML Height.

About Performance Considerations

When a chart is traversing records in the business component, its progress is indicated at the bottom of the window. Since traversing all of the records of a business component can be time-consuming, charts are not well suited for data sets larger than 1,000 records.

Various factors affect the performance of charts in Siebel applications:

- The number of records in the business component
- Whether the chart needs to search a multi-value group to obtain its data

- Whether a data point field is specified
- If the data point field is a currency field, the number of records whose currency is not the functional currency
- The processor, operating system, and database system

Using the Chart Applet Wizard

The following procedure identifies the steps required to create a new chart applet using the Chart Applet Wizard.

To create a new chart applet

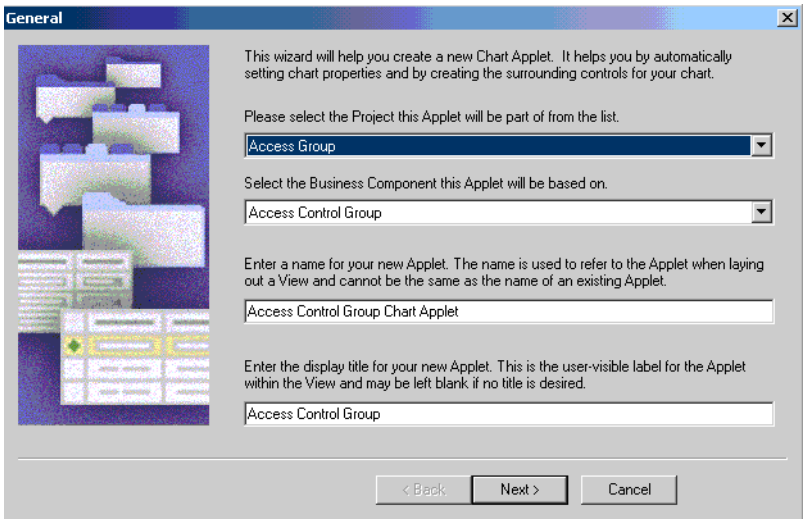
1. Choose File > New Object.

   The New Object Wizard dialog box appears with the list of objects that can be created through wizards.

2. Select the Applets tab and then select the Chart Applet icon.

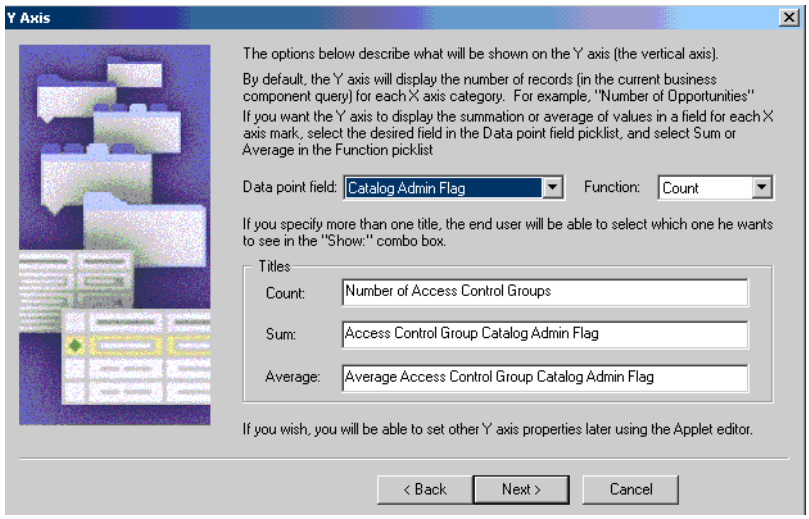
   The Chart Applet Wizard appears.

3. In the General dialog box, complete the following information and then click Next:
   - Project
   - Business Component
   - Name
   - Display Name



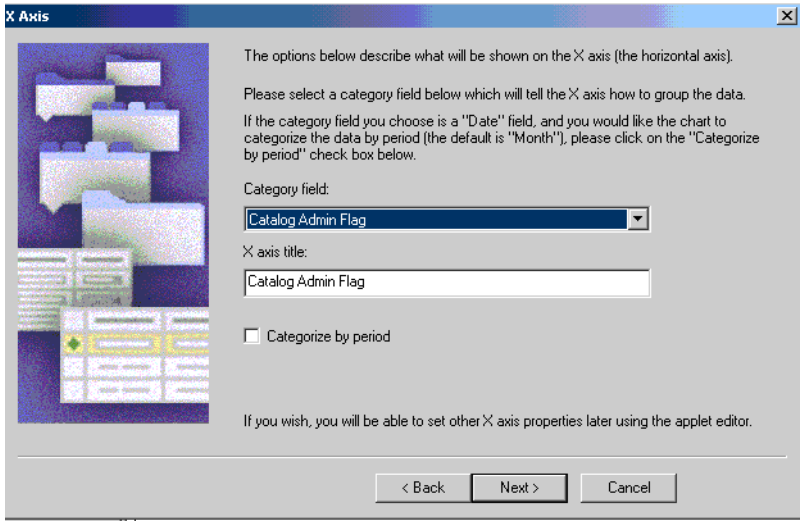
4 In the Y Axis dialog box, enter the options that will be displayed on the Y axis and then click Next.

For more information about the Y Axis, see "Axis Terminology" on page 421.



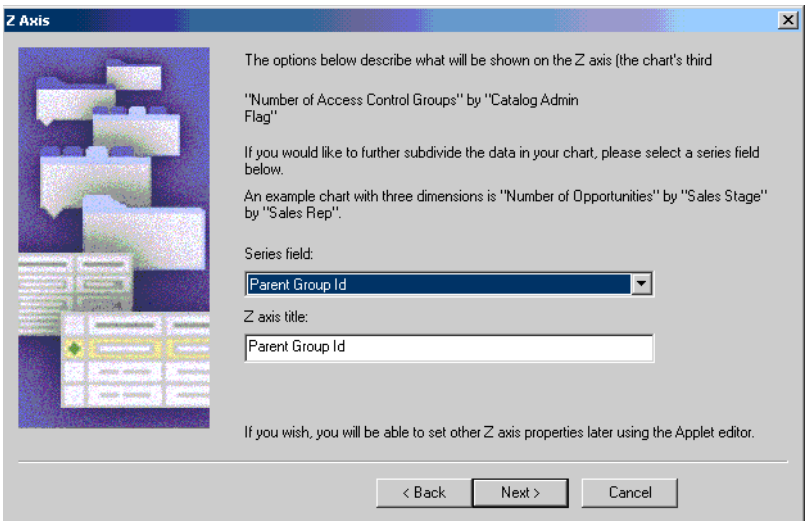
5 In the X Axis dialog box, enter the necessary information for the X Axis and then click Next.

For more information about the X Axis, see "Axis Terminology" on page 421.



6 In the Z Axis dialog box, enter the necessary information, and then Click Next.

For more information about the Z Axis see "Axis Terminology" on page 421.



7 In the Chart Title dialog box, enter a title, and then click Next.

8 In the Web-Layout-General dialog box, select the template that will be associated for the base read-only mode and then click Next.

9 In the Finish dialog box review the information and then click Finish.

The Chart Applet Wizard creates the necessary object definitions and sets the property values based on information you entered in the wizard. The Web Applet Layout Editor opens allowing you to map controls to placeholders in the Web template.

For more information, see "Editing the Layout of Web Page Objects" on page 325.

About Tree Applets

A tree applet is used to create a view, called an explorer view, that allows the user to navigate hierarchically through a structured list of records of related business components. An example of a tree applet and explorer view in Siebel Service is the Service Requests applet in the Service Request Explorer view, shown in Figure 119.

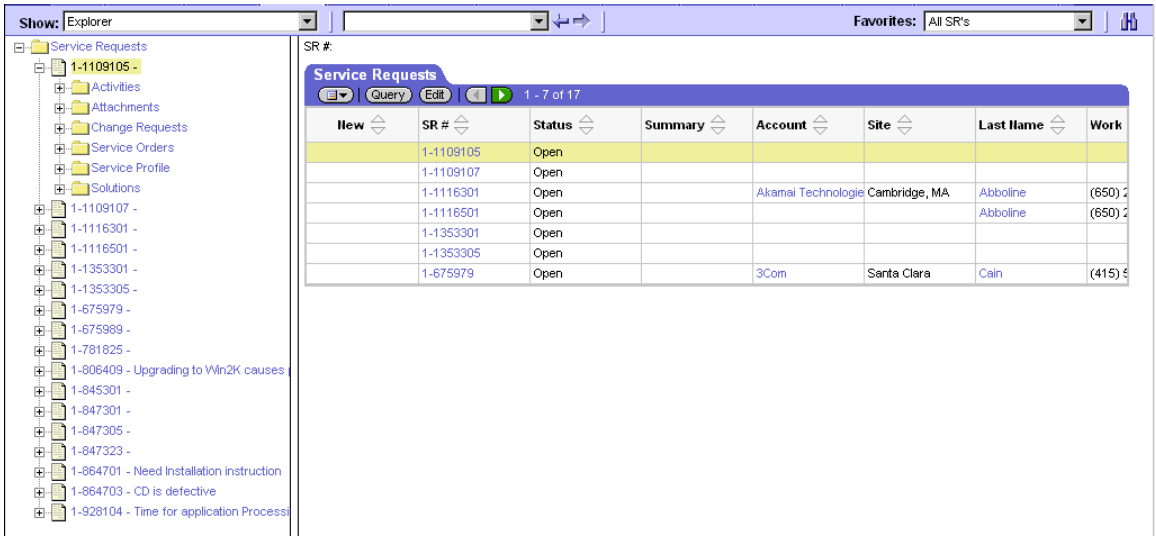


Figure 119. Service Requests Explorer View

This view (SR Explorer View) contains a tree applet (SR Tree Applet) in the left side, and one of various predefined list applets in the right side. The particular list applet that appears on the right depends on which node is selected in the tree on the left. For example, if the user double-clicks on the Change Requests folder in the tree hierarchy, the list applet on the right changes to display change requests records.

A tree applet in an explorer view is similar in operation to the Object Explorer and Object List Editor in Siebel Tools. The user may expand and collapse folders in the tree applet, and view the records in that folder in the list applet. The hierarchy displayed in the tree applet represents master-detail relationships between records of different business components.

For example, when the user expands a service request (document icon) by double-clicking, a set of folders appears hierarchically beneath it including Activities, Attachments, Change Requests, Solutions and so on. When the user expands one of these child folders, a list of records appears of the corresponding business component. If the user expands the folder for a service request, and then expands the Activities folder beneath it, the list of records displayed is the set of Activities for that service request. In the master-detail relationship between service requests and Activities, these Activity records are detail records of the master service request record that was expanded.

The user can also add or associate detail records of various kinds to particular master records. For example, the user could navigate through the hierarchy to the Solutions folder beneath a particular service request, click in the list applet, and select New Record from the applet-level menu to associate a solution record from an association applet. The product solution record would become a detail record of the service request.

A tree applet in an explorer view uses the set of master-detail relationships implemented in the business object assigned to the view. As described in "About Business Objects" on page 223, a business object implements a business model or entity-relationship diagram, and specifies the set of master-detail relationships between the business components it includes. This makes it possible to arrange the records of these various business components hierarchically, which can be a very useful feature.

Figure 120 shows the full set of master-detail relationships in the Service Request business object.

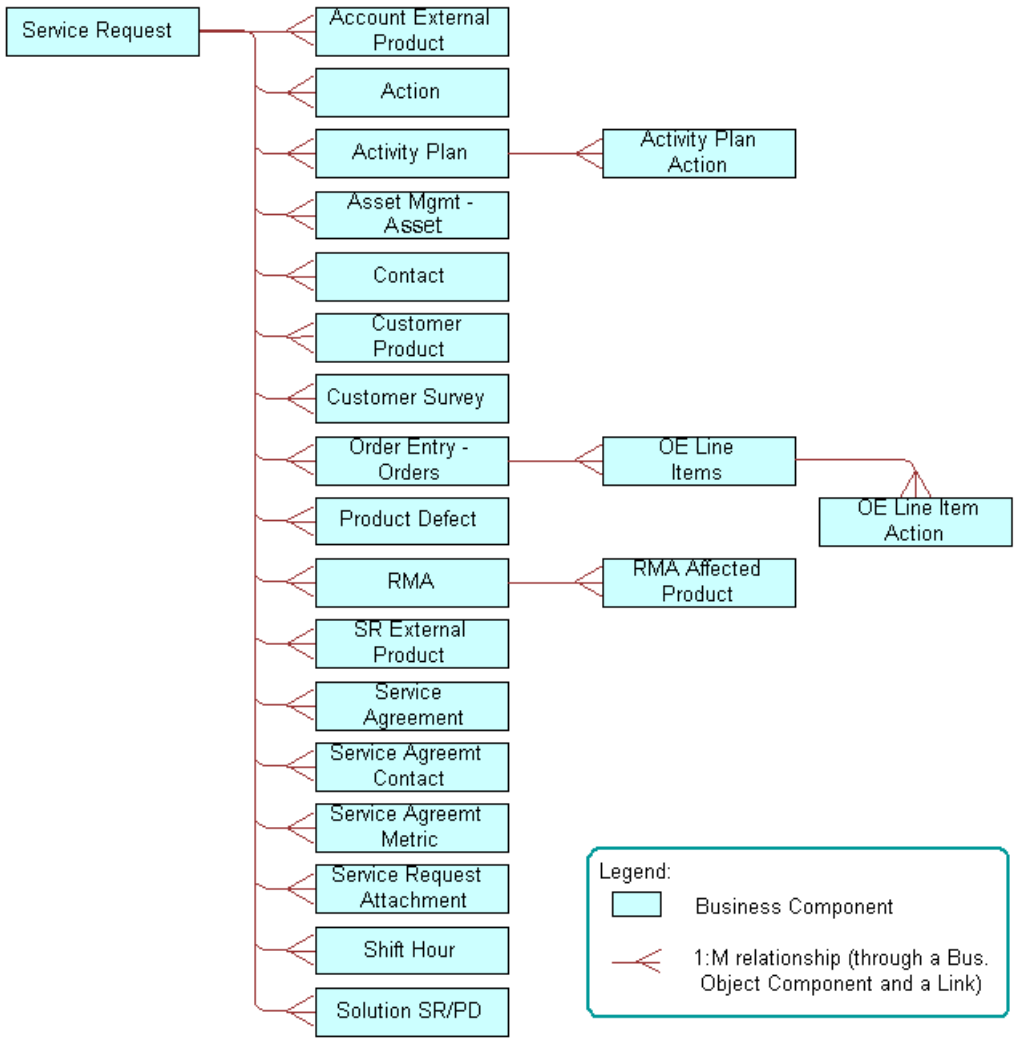


Figure 120. Service Request Business Object

The portion of the Service Request business object used in the Service Request Explorer view is shown in Figure 121.

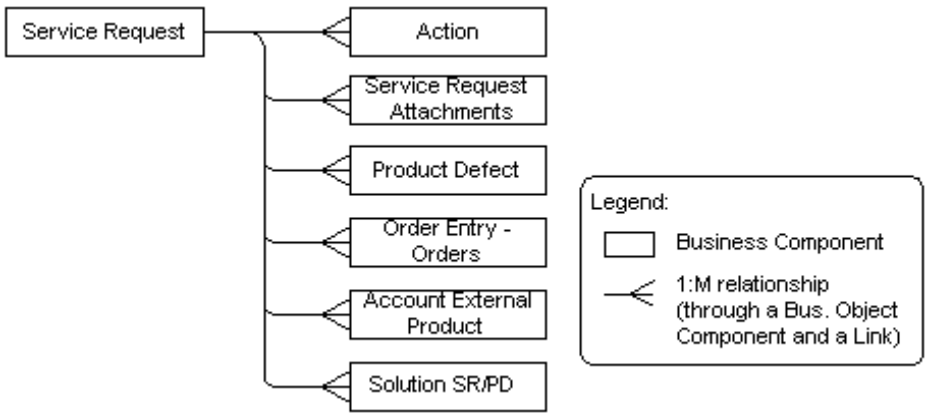


Figure 121. Service Request Business Object Components Used in SR Explorer View

The correspondence between business components in the Service Request business object and folder names in the tree applet is indicated in Table 61.

Table 61. Business Components Corresponding to Folder Names

Business Component	Folder Name in Tree Applet
Account External Product	Service Profile
Action	Activities
Order Entry - Orders	Service Orders
Product Defect	Change Request
Service Request	Service Requests
Service Request Attachment	Attachments
Solution SR/PD	Solutions

The tree applet and explorer view for service requests can be reconfigured to include additional business components. For example, Contacts, Customer Surveys, and Service Agreements folders could be added as child folders of Service Requests, and a Line Items folder could be added as a child of RMAs/Service Orders. However, only business components from the business object (Service Request in this case) can be added in an explorer view based on that business object. Furthermore, a business component can only be added as the immediate child folder of the business component that is its master in the business object. For example, you could add Order Entry Line Items as a child of RMAs/Service Orders, but not of Activities.

Configuring Tree Applets and Explorer Views

A tree applet appears in the left sectors of an explorer view. The applet has a tree object definition as a child. The tree object definition has tree node children. Each tree node child object definition implements one folder symbol. These object types are described in greater detail below.

View

An explorer view has a tree applet mapped to it as a View Web Template Item. However, there are no list applets mapped to it. The list applet is determined dynamically by the folder (Tree Node) that is currently highlighted by the user. A significant property of the View object the Business Object property. The business object selected determines which business components can be displayed, and which business components can be indicated as child nodes of which other nodes.

Tree Applet

A tree applet has no special property settings in the applet object definition, other than the class, which is set to CSSFrameTree. The Applet object type has the following important property settings:

- **Class.** Set to CSSFrameTree. This is required in order for the tree applet functionality to work.
- **Business Component.** Points to the same business component as the top-level tree node.

NOTE: Search specifications on tree applets are not supported.

Tree

The Tree object definition provides only a name; it is an object definition to which tree nodes can be attached, and which itself can be attached to the applet object definition. It always has the name "Tree." The Tree object type is similar to the List object type used in list applets, in that it serves as an attachment point for child object definitions.

Tree Node

Each folder symbol is implemented using one tree node object definition. This includes the top-level node (Service Requests in the example). All of the tree node object definitions are immediate child object definitions of the tree object definition. There is no hierarchy of child and grandchild tree node object definitions (reflecting the hierarchy in the tree applet) under the tree; this is not feasible in the object definitions hierarchy in the repository. Instead, each tree node's hierarchical position in the tree applet is specified in the Position property of the tree node object definition.

The Tree Node object definition has the following important properties:

- **Display Name.** This property specifies the name of the tree node (folder) as it will appear in the tree applet in Siebel applications. The display name appears to the immediate right of the folder symbol.

- **Applet.** This property specifies the applet that is opened in the right half of the view when the user opens the corresponding folder. Generally a list applet is specified. The applet must be based on a business component that is in the appropriate hierarchical position in the business object.
- **Position.** The tree node's hierarchical position relative to other tree nodes, and its sequence on its level, are specified with this property. The Position value consists of an integer, or a set of integers separated by periods, such as 1.1.2. The top-level node (Service Requests in the example) is specified as having a position of 1. All immediate child nodes of the top-level node have values of the form 1.*x*, where *x* specifies the node's order relative to other nodes on the same level. For example, in order for the Activities folder to appear after the Attachments folder rather than before it, their Position values (1.1 and 1.2, respectively) should be swapped.

  To attach a child node at the third level, you specify a Position value for the new node with its first two integers matching the position of the node to attach it to. For example, to attach a node to the RMAs/Service Orders node (currently 1.4), you would give the new node a position of 1.4.1. In general, the rightmost digit in a position specifies its order relative to others on the same level, and all other digits specify the position it attaches to.
- **Business Component.** This needs to be set to the same business component as is specified in the right-side applet.
- **Label Field.** This property points to the name of the field that is used to populate the names in the record list that appears when the node is expanded by the user. For example, the Order Number field would provide the values for the RMAs/Service Orders node, and the Description field for the Activities node.
- **Selected Bitmap Index.** This should be set to the value 5, which corresponds to the folder symbol.

Using the Tree Applet Wizard

The following procedure identifies the steps required to create a new tree applet using the Tree Applet Wizard.

NOTE: The Tree Applet Wizard creates the tree object but does not create Tree Node child objects. You must add a Tree Node object for each applet that you want to appear in the Explorer section of the view, including the top-level node, such as service requests. For more information, see "Tree Node" on page 451.

To create a tree applet using the Tree Applet Wizard

1. From the Tools main menu, choose File > New Object.

   The New Object Wizard dialog box appears.
2. Select the Applets tab and then select the Tree Applet icon.
3. In the General dialog box, enter the following information, and then click Next:
   - Project
   - Business Component

- Name
- Display Name

4 In the Web Layout-General dialog box, select the Web template to use for the tree applet, and then click Next.

   Some templates used for tree applets are:

   - Applet Tree
   - Applet Tree 2
   - Applet Tree Marketing

5 In the Finish dialog box, review the information, and then click Finish.

   The Tree Applet Wizard creates the tree object and sets the required properties based on the information you entered.

Creating Tree Applets in the Applet Layout Editor

Tree applets can be created and modified in the Web Layout Editor. When you drag a TreeControl onto the applet, the tree controls and the Tree object definition are created.

When you right-click over the tree control, a pop-up menu appears with the following tree-specific options:

- **Select Tree option.** Allows you to copy and paste the tree control into another applet.
- **Create New Tree Node option.** Adds a new tree node to the tree. The tree node is created at the top level, and is subsequently moved using the Move Selected Tree Node option.
- **Move Selected Tree Node option.** Allows you to change the position of the tree node in the tree. You first click on the tree node you wish to move. Then you can use up, down, left, and right arrow keys, with the SHIFT key depressed, to move the node up or down a level or change its position within its level.

The Position property on all of the nodes is automatically updated for all operations.

Pressing the DELETE key when the tree appears in the Applet Web Template Layout window deletes the currently selected tree node. The Undo and Redo options in the Edit menu are active for all tree manipulation operations in the Applet Web Editor.

About Recursive Trees

In a recursive tree, all levels in the hierarchy are of the same object type. For example, the Account Explorer Applet consists of a tree applet in which the only node is for the Account business component, and subaccounts appear beneath accounts which have them. Recursive trees are provided in standard Siebel applications for accounts, activities, campaigns, opportunities, positions, and various other business components in which records can have subrecords. Almost any number of levels of subrecords are possible in a recursive tree.

For a recursive tree to be implemented, the business component used must contain a pointer to the record of the same type at the next level up in the hierarchy. In the accounts tree example, the Account business component has a Parent Account Id field which points to its parent account. A Link object definition must exist that references this field in its Destination Field property. In the accounts example, this link is Account/Account.

A recursive tree is implemented with a tree object definition to which only one tree node object definition is attached. In the Tree Node object definition, the following special properties are set:

- **Recursive.** This property is set to TRUE to indicate that this is a recursive tree.
- **Recursive Link.** This property points to the link object definition that specifies the one-to-many relationship between the master business component and itself.
- **Root Search Spec.** This property contains a search specification expression that identifies how the list of top-level records is derived. Generally the top-level records are those that have nothing in the parent Id field, and therefore the search specification is of the form "[Parent *xxx* Id] is NULL," for example, "[Parent Account Id] is NULL."

File Attachment Applets

A *file attachment applet* (or *attachment applet*) provides access to external documents, such as spreadsheets, word processing documents, and slide presentations, from within Siebel applications. A file attachment applet provides the following capabilities:

- Allows the user to open a document of any Windows-supported file type by clicking on its name in a list.
- Allows the user to add document files to a list, edit them, or remove them.
- Provides synchronization and shared access support for attached documents.

An example of a file attachment applet appears in Figure 122.

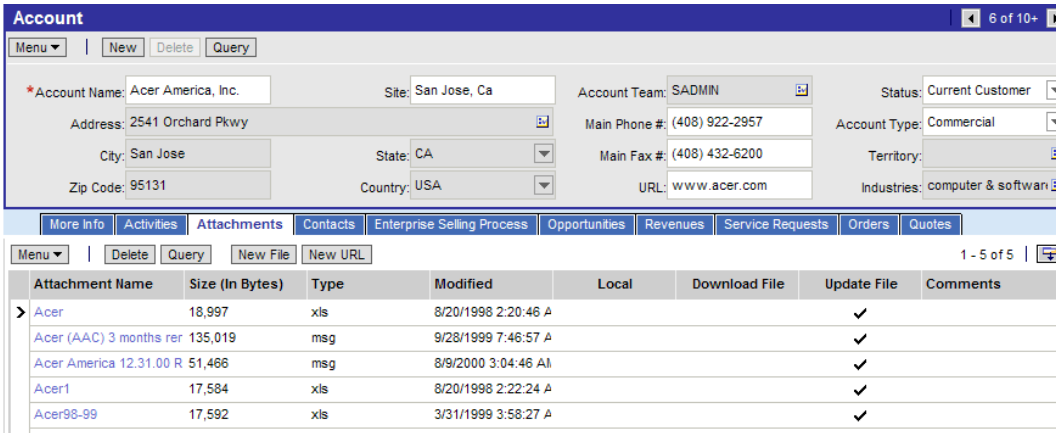


Figure 122. File Attachment Applet: Account Attachment View

Figure 122 shows the Account Attachment view. The upper applet is the standard Account Form Applet. The lower applet is a file attachment applet called Account Attachment Applet. There is a master-detail relationship between the account and the list of account attachments, so that all file attachments for the current account are listed in the lower applet.

Each document is represented by a row in the attachments list. The document's filename, local/server status, file size, Windows file type (filename extension), and date of last update are displayed. Additionally, the name of each file appears in underlined, colored text, indicating that the file may be opened in the appropriate Windows application by clicking on the name.

Users add documents to the attachment list by choosing the New File button or selecting Menu > New Record, and then clicking the select button in the Attachment Name field. The Siebel application searches for files to be attached in the directory specified in the Start in field of the program icon's Properties dialog (Start >Programs > *Program_Name* > *Program_Icon*). If a user chooses a different folder when attaching a file, the Siebel application searches for the file in this folder the next time the user attaches a file.

Configuring Attachment Applets

Attachment applets use functionality built into the Siebel file system. Various specialized objects and methods are provided in the Siebel File System that provide the attachment support and synchronization capabilities:

- An attachment applet is based on an attachment business component. The details of attachment business components are discussed in the section "Configuring Attachment Business Components" on page 456.
- An attachment applet has the Class property set to either CSSFrameListFile or CSSFrameFile. CSSFrameListFile is used for attachment list applets. CSSFrameFile is used for attachment form applets.

- The Name list column or text box control has a Detail Applet property setting of File Popup Applet. This refers to the dialog box that appears when you click on the ellipsis button in the list column or text box.

Several of the list columns or controls in the applet are based on corresponding fields in the attachment business component. This business component is described in "Configuring Attachment Business Components" on page 456. These will typically include those listed in Table 62.

Table 62. List Columns or Controls in an Attachment Applet

Display Name	Field	Type
Name	*xxx*FileName	TextBox
Local	Dock Status	CheckBox
Request	*xxx*FileDockReqFlg	CheckBox
Size	*xxx*FileSize	TextBox
Type	*xxx*FileExt	TextBox
Modified	*xxx*FileDate	TextBox
Auto Update	*xxx*FileAutoUpdFlg	CheckBox

The "*xxx*" prefix refers to a standard prefix found in the names of the fields in the attachment business component. For example, for account attachments the prefix is "Accnt" and the actual field names referenced from the applet would be AccntFileName, AccntFileDockReqFlg, and so on.

Configuring Attachment Business Components

The Business Component property of the attachment list applet identifies the business component that the Siebel file system uses to store the attachment list data. For the Account Attachment Applet, this business component is called Account Attachment. The attachment business component must adhere to the following requirements:

- The Class property of the Business Component object must be set to CSSBCFile.
- The Table property must refer to an attachment table, as described in the section "Configuring Attachment Tables" on page 458. In the Account Attachment Applet, this table is S_ACCNT_ATT.
- Two Business Component User Prop object definitions must be created as children of the attachment business component, as follows:

- **DefaultPrefix user.** This is the text of the prefix used in the names of the Siebel File Engine-based field object definitions. These are fields which are based on the base table for the business component. (There may be fields which are based on a joined table, and these will have a different prefix.) For the Account Attachment business component in the example, the prefix is "Accnt," which appears in field names such as AccntFileName and AccntAutoUpdFlag.
- **FileMustExist user.** This is a TRUE/FALSE value indicating whether or not the user can enter the name of a file to be provided later. Typically this is set to TRUE, indicating that the file must already exist in order to add it as an attachment.

- The FileDockReqFlg field must have the value for the predefault property set to "N". The FielDockReqFlg maps to the required column FILE_DOCK_REQ_FLG in the attachment table.

The field names of file engine-supplied fields have to adhere to a special format, and map to specific column names in the attachment table. These names consist of the prefix, as specified in the DefaultPrefix user property, followed by a required set of suffixes. These field names, corresponding columns, and data types are listed in the Table 63.

Table 63. Fields in an Attachment Business Component

Name	Column	Type	Text Length
*xxx*FileAutoUpdFlg	FILE_AUTO_UPD_FLG	DTYPE_BOOL	1
*xxx*FileDate	FILE_DATE	DTYPE_DATETIME	
*xxx*FileDeferFlg	FILE_DEFER_FLG	DTYPE_TEXT	1
*xxx*FileDockReqFlg	FILE_DOCK_REQ_FLG	DTYPE_TEXT	1
*xxx*FileDockStatFlg	FILE_DOCK_STAT_FLG	DTYPE_TEXT	1
*xxx*FileExt	FILE_EXT	DTYPE_TEXT	10
*xxx*FileName	FILE_NAME	DTYPE_TEXT	220
*xxx*FileRev	FILE_REV_NUM	DTYPE_ID	15
*xxx*FileSize	FILE_SIZE	DTYPE_NUMBER	
*xxx*FileSrcPath	FILE_SRC_PATH	DTYPE_TEXT	220
*xxx*FileSrcType	FILE_SRC_TYPE	DTYPE_TEXT	30

Table 64 lists a non-file engine field that will usually be present, although it is not required.

Table 64. Non-File Engine Field in an Attachment Business Component

Name	Column	Type	Calculation
Dock Status	(calculated)	DTYPE_BOOL	IIf ([AccntFileDockStatFlg] = "N" OR [AccntFileDockStatFlg] IS NULL,"N","Y")

Additional fields can be included as needed. For specialized uses of attachments, such as an image control, the file engine fields may be present in addition to the fields from a standard business component (often through a join). For example, a Product or Literature business component can contain file engine fields to support display of product picture or brochure picture bitmap images.

Multiple sets of file engine fields from different tables can be incorporated in the same business component. For example, literature attachments can have subattachments, with the subattachments derived from an intersection table or extension table. The field name prefix must be different for each table.

Configuring Attachment Tables

Attachment tables provide the underlying data storage for the attachment business components. Unlike the attachment business component, which can support purposes in addition to file engine functionality, the attachment table stores file engine data only.

Users will not populate the attachment table directly. Rather, users are provided with an initially empty attachment table, and populate it one file at a time using drag and drop or the browser dialog box in the corresponding file attachment applet.

Table 65 lists the columns that appear in an attachment table. Note that the columns whose names begin with FILE_ are required columns, and must be named as specified in the table. The User Name values can be the same as or different from those listed.

Table 65. File Columns in an Attachment Table

Name	Default	User Name	Type	Physical Type	Length
FILE_AUTO_UPD_FLG		File Auto Upd Flg	Data (Public)	Character	1
FILE_DATE		File Date	Data (Public)	Date Time	
FILE_DEFER_FLG		File Defer Flg	Data (Public)	Character	1
FILE_DOCK_REQ_FLG		File Dock Req Flg	Data (Public)	Character	1
FILE_DOCK_STAT_FLG		File Dock Stat Flg	Data (Public)	Character	1
FILE_EXT		File Ext	Data (Public)	Varchar	10
FILE_NAME		File Name	Data (Public)	Varchar	255
FILE_REV_NUM	0	File Rev Num	Data (Public)	Varchar	15
FILE_SIZE		File Size	Data (Public)	Number	22
FILE_SRC_PATH		File Src Path	Data (Public)	Varchar	255
FILE_SRC_TYPE		File Src Type	Data (Public)	Varchar	30

Various system columns not related to the file engine will also be present, such as CREATED, LAST_UPD_BY, and ROW_ID.

A table that has file engine columns must be flagged as such with a TRUE value in the File property of the corresponding table object definition.

Pop-Up Windows

This section describes how to create pop-up windows and dialog boxes. There are various scenarios in which pop-up windows are implemented:

- "Configuring Pop-Up Applets Launched from Applets"
- "Configuring Pop-Up Wizards" on page 460
- "Configuring Pop-Up Views Launched from Applets" on page 461

When configuring pop-up windows, consider the following:

- Pop-up applets must use classes derived from CSSSWEFramePopup. Business components are not required for the applets. However, if your pop-up applet is associated with a business component, that business component must be a child of the business object of the view containing the applet from which the pop-up window is launched.
- One level of pop-up window is supported. If you activate a pop-up window from within a pop-up window, it replaces the original pop-up window.
- The More/Less feature is not supported on Pop-up applets. See "Configuring Controls and List Columns to Appear in More Mode Only" on page 277 for more information.

Configuring Pop-Up Applets Launched from Applets

This is the typical scenario in which clicking a button on an applet invokes a pop-up window for editing a set of values, or browsing through a list, and so on.

To configure a pop-up applet

1. Select the applet from which to launch the pop-up window.
2. Create a control for the applet.
3. Set the Method Invoked property of the control to ShowPopup.
4. Expand the Control object, and then select the Control User Prop object.
5. Create three control user properties:
   - **Popup.** Set to the applet you want to appear. This applet must use a class derived from CSSSWEFramePopup.
   - **Mode.** Optional. Mode of the applet, either Base or Edit. If not specified, the default is Base.

- **Popup Dimension**. Optional. Dimension of the pop-up window. The format is Height X Width, for example 500 X 800. If not specified, the dimensions will default to the value specified in the applet's HTML Popup Dimension property. If that is not specified, the pop-up window dimensions will default to 600 X 600.

6 Create the pop-up applet.

7 Add controls to the pop-up applet:

- **Cancel.** Set the Method Invoked property of the control to CloseApplet or UndoRecord. This will close the pop-up applet when Cancel is clicked.
- **OK.** Set the Method Invoked property of the control to CloseApplet to close the applet after you finish processing other calls within your invoked method. This will close the pop-up applet, and then refresh the parent applet in the main browser window.

Configuring Pop-Up Wizards

If you want to have a wizard-style set of pop-up applets, the procedure is similar to configuring a dialog box invoked from an applet control. See "Configuring Pop-Up Applets Launched from Applets" on page 459.

NOTE: The parent applet must be in Edit mode.

To configure a pop-up wizard

1 In Siebel Tools, select the pop-up applet, expand the Applet object, and then select the Applet Web Template object.

2 Add multiple templates of type Edit in Siebel Tools to the Applet Web Template.

3 Assign a different value to the Sequence property of each of the templates, in the order you want them to appear.

4 To navigate between pages, add two controls:

a Previous button:

- Set the Method Invoked property to PostChanges.
- Add a Control User Prop child object called Sequence with a value of -1.

This posts the changes that the user has made, and then goes back to the page whose sequence number is one less than the current one.

b Next button:

- Set the Method Invoked property to PostChanges.
- Add a Control User Prop child object called Sequence with a value of 1.

This posts the changes that the user has made, and then goes to the page whose sequence number is one greater than the current one.

5 On the last template, create a control called Finish that closes the applet, and then updates the parent applet.

Configuring Pop-Up Views Launched from Applets

A view (rather than a single applet) can be loaded into a pop-up window. This is not recommended, especially in employee applications. Instead, you should navigate to a new view in the same window (invoking the GotoView method) rather than pop up a view in a new window. However, if there is a requirement for this functionality, perform the following task.

To configure a pop-up view

1. Select the applet from which to launch the pop-up view.
2. Create a control for the applet.
3. Set the Method Invoked property of the control to ShowPopup.
4. Expand the Control object, and then select the Control User Prop object.
5. Create two control user properties:
   - **View.** Set to the view that you want to pop up.
   - **Popup Dimension.** Set the dimensions of the pop-up. The format is Height X Width, for example 500 X 800.

Users must use the browser's Close (X) button to close the window. There is no way to close the window programmatically.

21 Configuring Special Purpose Controls

Topics in This Chapter

ActiveX Controls

An ActiveX control is a self-contained program unit that can be run from within other programs. An ActiveX control typically registers itself in the Windows registry. In Siebel applications, any registered ActiveX control can be incorporated in an applet. This provides the means to add one or more specialized features to an applet, such as a slider or media player. You can also embed entire applications that are available as ActiveX controls.

NOTE: ActiveX controls will work in most environments, but the programming environment itself may or may not support it. For example, trying to insert a Siebel ActiveX application control into an Excel worksheet generates a "Cannot insert object" error.

Certain third-party ActiveX controls, for example Microsoft Web Browser, Microsoft Rich Textbox, and CTreeView, do not work with Siebel applications and are not supported.

Creating DLL and Class Objects That Reference an ActiveX Control

To make an ActiveX control available for use, you must create DLL and Class objects in Siebel Tools that reference the CAB file containing the control.

To create an ActiveX control

1. Create a CAB file containing the control, if it does not already exist.

   Microsoft provides some utilities for doing this.

2. Copy the CAB file to the correct folder.

a When deploying to a server environment, copy the CAB file to the <*Siebel_install_dir*>\SWEApp\public*language_code*\applets folder,

where:

<*Siebel_install_dir*> is the full path to the Siebel Web Applications installation directory on the Siebel Web Server

language_code is the three-letter code for the language, for example ENU for US English and JPN for Japanese

b When deploying a CAB file on the Mobile Web Client, copy the CAB file to the <*Siebel_Client_install_dir*>\PUBLIC*language_code*\applets folder,

where:

<*Siebel_Client_install_dir*> is the full path to the Siebel Mobile Web Client root directory

language_code is the three-letter code for the language, for example ENU for US English and JPN for Japanese

3 In Siebel Tools, select the DLL object, and then add a new record.

4 Fill in the fields as shown in the following table.

Field	Value
Name	User-defined name for the DLL object
Project	A currently locked project in the Siebel Repository
File Name	File name & version that references the CAB file containing the ActiveX control, for example: subman.cab#Version=7,0,0,0
Code or Class Id	Class Id of the ActiveX control, for example: clsid:06314967-EECF-11D2-9D64-0000949887BE

5 Select the Class object, and then add a new record.

6 Fill in the fields as shown in the following table.

Field	Value
Name	User-defined name of the Class object
Project	The locked project used in Step 4
DLL	Name of the DLL object created in Step 4
Object Type	ActiveX Control

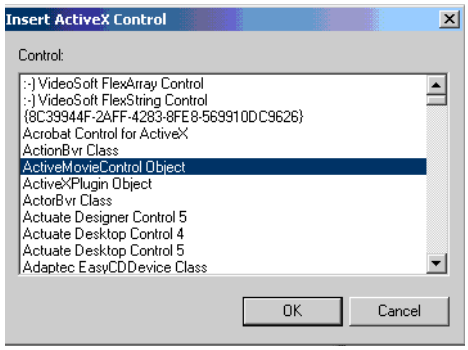
Adding an ActiveX Control to an Applet

You add an ActiveX control to an applet using the Applet Layout Editor.

To add an ActiveX control to the applet

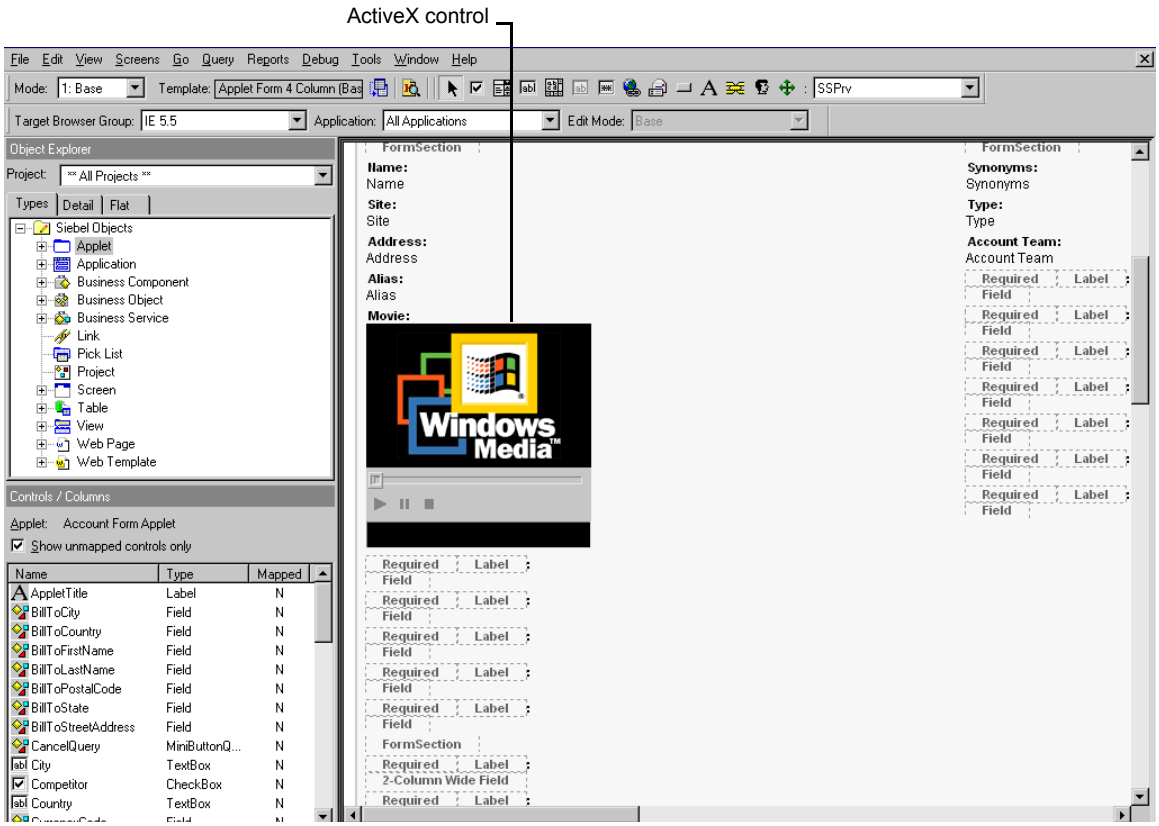
1. Open the applet in the Applet Web Editor.
2. Click the ActiveX Control toolbar icon in the Web Control toolbar.
3. Drag the control to a placeholder in the Web template.

   The Insert ActiveX Control dialog box appears for the selection of one of the currently registered ActiveX controls on your system.



4 Select the desired ActiveX control and click OK.

The selected control replaces the placeholder in the Applet Web Editor.



The illustration shows a movie player control called ActiveMovie Control in a form applet.

5 Set the Class property of the control to the name of the class created in Step 6 of "To create an ActiveX control."

The default properties of the ActiveX control will be defined as Control User Properties.

6 Compile the .srf file and test.

Setting Properties in an ActiveX Control

An ActiveX control includes its own property list, which varies from control to control. In addition, an ActiveX control in an applet has the full set of properties of the Control object type. There are two ways to view and modify properties for an ActiveX control in an applet: by using the Properties window, or by activating the control's built-in property sheet.

To change properties in the Properties window

1 Choose View > Windows Properties.

2 Select the ActiveX control in the Applet Web Template Layout window.

   The Properties window lists the properties for the ActiveX control.

3 Click the Categorized tab at the top of the Properties window.

   This clusters all of the control's native properties under the ActiveX heading, and all of the standard Control object type properties under the Misc. heading.

4 Make changes to property settings as you would in any Siebel object definition.

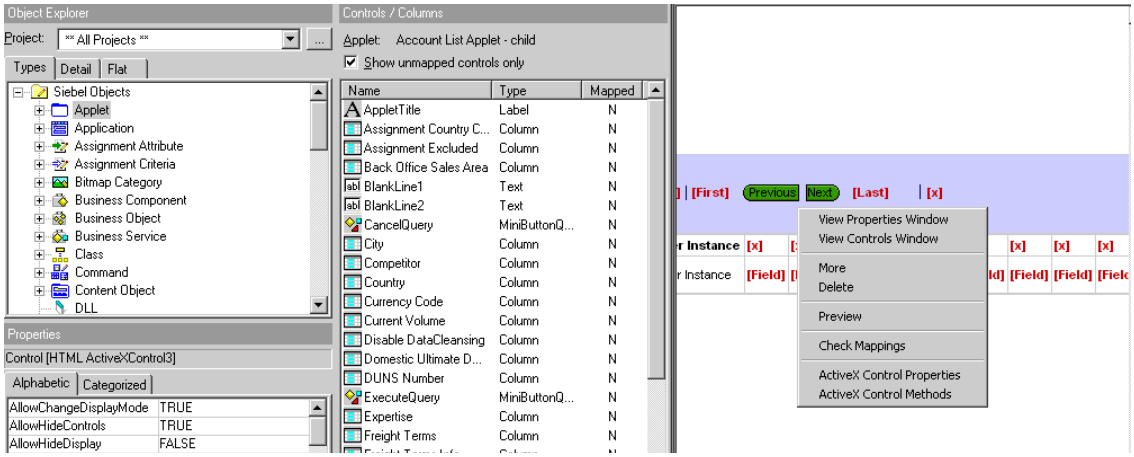
The changes you make to the control's native properties are generally displayed in the Applet Web Template Layout window, such as when you change a text color or font property. Additionally, changes you make to the control's native properties are saved with the applet, just as with the Siebel properties.

The alternative approach to changing property settings is to use the control's native property sheet. When you make property changes using the control's native property sheet, you can modify only the properties of the ActiveX object, not those of the standard control object definition. To alter the standard control properties, you must use the Properties window.

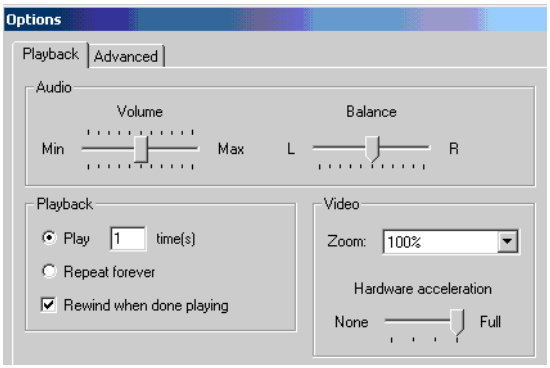
To change properties in the ActiveX control's native property sheet

1 Select the ActiveX control in the Applet Web Template Layout window, and right-click.

2 In the shortcut menu that appears, select ActiveX Control Properties.



If the control has a native property sheet, it is activated. The native property sheet for the ActiveMovie control appears below:



3 Make any desired changes to property settings. These settings are saved with the applet when you exit the Applet Web Template Layout window or do a Save.

ActiveX Methods and Events

An ActiveX control in an applet exposes a set of methods and events that are provided with the control. The methods may be called from scripts written in browser script attached to the control or other objects, and event procedures can be programmed in to respond to the events the control generates.

To see the list of available methods for the ActiveX control

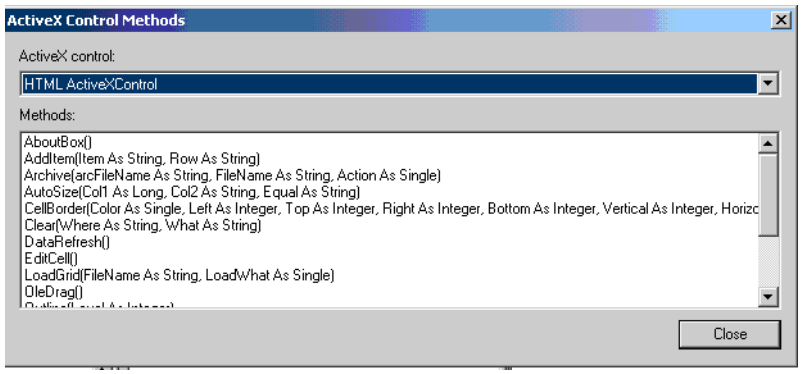
1 In the Applet Web Template Layout window, click the ActiveX control to select it.

2 Right-click to access the shortcut menu.

See "To change properties in the ActiveX control's native property sheet" on page 467 for an illustration of the shortcut menu.

3 Select the ActiveX Control Methods menu option.

The ActiveX Control Methods dialog box appears.



This dialog box lists the methods, and specifies the syntax for calling them. It is for reference purposes only.

4 Click Close to dismiss the dialog box.

Distributing ActiveX Controls

You can distribute ActiveX controls to licensed end users. When you do this, you will also need to distribute any dependency DLLs along with the ActiveX controls. For information about these dependency DLLs (such as the number and type of DLLs that need to accompany the ActiveX controls), contact the ActiveX vendor.

HTML Content Controls

HTML content controls allow you to display HTML content in fields in the user interface. The HTML content can be static HTML or HTML from an external content source.

Configuring fields to display HTML content is a matter of setting properties of the control in Siebel Tools and defining any necessary supporting information, such as URL, host name, syntax, and authentication parameters using a set of administration views in the Siebel Web client.

Control Properties

The key Control object type properties for displaying HTML content are the following:

- **Field Retrieval Type.** This property determines the type of HTML to be displayed in the field. Possible values are:

- **Symbolic URL.** This value specifies that the content will come from an external host based on a symbolic URL. You need to define the necessary information needed to access the external source. This includes the syntax used for the request, the host name, necessary arguments, and so on.

  For more information about defining symbolic URLs, see *Siebel Portal Framework Guide*.

- **Field Data.** This value specifies that the HTML content will be stored as data.

- **Service.** This value specifies that the field will be rendered by a business service. The control must have a User Prop defined with the name *Field Retrieval Service* and the value is the name of the business service.

  For example, you can define a control to display a Content Center asset by setting the Field Retrieval Type to Service and then adding a Control User Property child object with the name Field Retrieval Service and the value ContentBase - Asset Publish Service.

  For more information about Content Center Assets, see *Applications Administration Guide*.

- **HTML Attachment.** This value specifies that the field will display an HTML attachment. The control will render the HTML Attachment identified by the underlying field.

- **URL.** This value specifies that content will be displayed from an external source based on the simple URL specified in the underlying field.

- **ContentFixupName.** This property determines how to correct links post processing. It provides the name of a Fixup as displayed in the Fixup Administration View. This value does not work if Field Retrieval Type is HTML Attachment or Service.

- **HTML Display Mode.** This property should be set to DontEncodeData so that the HTML content renders properly in the browser. Possible values are:

  - **DontEncodeData.** Use this value when the field value is actual HTML text and you want it to be shown as such.

  - **EncodeData.** If the field value contains HTML reserved characters, such as angle brackets (< >), ampersand (&), and so on, they are encoded before they are displayed so that they appear correctly within the browser.

Administration Views

There are four administration views that allow you to specify the necessary information related to HTML content controls. This includes information such as host name, URL, required arguments, authentication parameters, and so on.

The views used to enter this information are described in Table 66:

Table 66. Administration Views for HTML Content

View Name	Menu Path	Description
Host Administration	From the application-level menu, choose Navigation > Site Map > Administration - Integration > WI - Symbolic URL List > Host Administration.	Allows you to enter the HTTP host, including virtual name and authentication parameters.
Fixup Administration	From the application-level menu, choose Navigation > Site Map > Administration - Integration > WI - Symbolic URL List > Fixup Administration.	Allows you to specify how to handle links post processing.
Symbolic URL Administration	From the application-level menu, choose Navigation > Site Map > Administration - Integration > WI - Symbolic URL List > Symbolic URL Administration.	Allows you to specify the content agent for an external host. This includes URL, host name, fixup name, and arguments.
Content Sets	From the application-level menu, choose Navigation > Site Map > Administration - Content Center > Content Sets.	Allows you to upload and manage Web content to be rendered in the Siebel application.

For an overview of content agents and symbolic URLs as well as procedures for administering content agents, see *Siebel Portal Framework Guide*.

The Host Administration View

The Host Administration view is used to specify hosts that either require fixup processing, authentication, or to simply obscure the true host name. Only links associated with a specified host are fixed up.

For each host, you need to specify an external content host server. The specification of a host enables one or more of the following features:

- It obscures the true servername in the generated HTML.
- It allows the specification of a set of NCSA Basic Authentication credentials for content hosts that require authentication.
- It allows administrators to control fixup at the host level.

Fixup Administration View

Administrators can use this view to control the behavior of links embedded within the external content. A fixup has a Link Context, which corresponds to the fixup type. There are four types of fixups:

- **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 from the Hosts table (for example, navigated to by choosing Integration Administration > Host Administration) are proxied in order to maintain all of the SWE context.
- **Inside Applet.** This fixup performs the same as the Inside Application fixup.

NOTE: Fixup is required for all links within applications that use high interactivity.

Proxied Links

When using either the Inside Application or Inside Applet fixup type, any link using a host from the Hosts table is proxied. Any relative link is first converted to an absolute URL, and then if the host is in the Hosts table, the link is proxied.

Default Link Targets

There are no default link targets applied to a fixup. However, a fixup may have a link target specified for it, in which case the link target will be added to the fixup.

Configuring Fields to Use Web Content Assets

Any business component can take advantage of Web Content Assets to add fields that are rendered as HTML content. For example, you can use display static HTML messages to your end users in the Partner Relationship Manager application, or represent a Product Description as HTML content.

This section describes the steps for configuring this functionality, using the Partner Message business component as an example.

To configure fields to use Web Content Assets

1. In Siebel Tools navigate to the Partner Message business component.
2. Navigate to the Message field and set the Pick List to ContentBase Asset Hierarchical PickList.
3. Query for Partner Message List Applet.
4. Navigate to the list column named Message Body and set the Pick Applet property to ContentBase Asset Hierarchical PickList.

5 Query for Partner Message Entry Form Applet.

6 Navigate to the control name Message Body Preview and set the Field Retrieval Type property to Service.

7 Navigate to the Control User Props for this control and add a Control User Prop with the name Field Retrieval Service and the value ContentBase - Asset Publish Service.

8 Query for Partner Message Form Applet (SCW).

9 Navigate to the control named MessageBody and set the Field Retrieval Type property to Service.

10 Navigate to the Control User Props child object for this control and add a control user property with the name Field Retrieval Service and the value ContentBase - Asset Publish Service.

11 Compile your changes to an .srf file and test.

22 Displaying Images

Topics in This Chapter

About Displaying Images in the Siebel User Interface

Several object types allow you to display images in the Siebel user interface. The object types include the following:

- **Bitmap Categories.** The Bitmap Category object type allows you to group image files together by function. For example, there is a category called Button Icons that contains all images for buttons on applets used in Siebel applications.
- **Bitmap Objects.** Defines images, such as JPG and GIF, that are referenced by UI objects such as buttons and controls.
- **Icon Maps and Icons.** Allow you to display images for field values.

  HTML Hierarchy Bitmaps. Allow you to display images in hierarchical applets, such as tree applets.

You use a combination of these objects for tasks such as:

- "Creating Bitmap Categories and Bitmap Objects" on page 476
- "" on page 477
- "Defining Images Used in Hierarchical Objects" on page 479
- "Using Images as Links in Controls" on page 480

Creating Bitmap Categories and Bitmap Objects

Bitmap objects allow you associate image files, such as GIF and JPG, with Siebel objects, such as button controls and fields. Images are defined in the repository using the Bitmap object type. They can be of any format supported by the target browser. The Bitmap object type identifies the location of the image file and other properties, such as width and height.

Image files are handled differently, depending on the file type:

- Images of type BMP are imported into the repository and the File Name field of the Bitmap Object becomes read-only.
- Image files such as GIF and JPG are not stored in the repository; instead they are stored in the Public\$lang\images folder of your Siebel installation and referred to from the repository Bitmap objects.

NOTE: Only images that are associated with Siebel objects, such as icon maps, page tabs, and so on, are defined as Bitmap objects in the Siebel repository. Some images used in Web templates, such as static images, are not associated with Siebel objects and are not defined as Bitmap objects in the Siebel repository. These objects are defined in the application's configuration (.cfg) file.

The Siebel Web engine (SWE) renders the bitmap object using the HTML <img> tag. The *Height* and *Width* attributes of the Bitmap object can be set to the height and width of the image that you want to display on the Web page. If these attributes are set, SWE uses them as "width" and "height" attributes of the <img> tag. This allows the creation of various bitmap objects that share the same image file, but are rendered with different dimensions. The Bitmap object has another new attribute called *Alt Text*. This attribute can be set to the text to be used in the "alt" attribute of the image tag. The other attributes of the Bitmap Object like *Data* and *Transparent Color* are not used with Web Images.

To create a bitmap objects for image files of type BMP

1. Navigate to the Bitmap Category object type and create a new bitmap category or select and existing Bitmap Category.
2. Select the child Bitmap object.
3. Create a new bitmap object and then use the information in the following table to define the bitmap object properties:

Property	Description
Name	Name of the bitmap object definition.
Alt Text	Alternative text used in place of name property for a bitmap.
Height	The height (in pixels) of the bitmap.
Width	The width (in pixels) of the bitmap.

4 With the new record selected, right-click, and then choose Import Bitmap.

The Open Dialog box appears.

5 Use the Open Dialog box to navigate to the BMP file that you want to import.

The BMP file is imported into the Siebel repository.

To create bitmap objects for image files of type GIF

1 Navigate to the Bitmap Category object type and create a new bitmap category or select and existing Bitmap Category.

2 Select the child Bitmap object.

3 Create a new bitmap object and then use the information in the following table to define the bitmap object properties:

Property	Description
Name	Name of the bitmap object definition.
Alt Text	Alternative text used in place of name property for an image.
File Name	Name of the image file. For images that are published within subfolders in the image folder, include the subfolder in the image name. For example, for an image named *asterix.gif* that is published in the `eapps/public/enu/images` folder, set the *File Name* attribute to `asterix.gif`. However, for the image `next_on.gif` that is published in the `eapps/public/enu/images/bttns` folder, set the *File Name* attribute to `bttns/next_on.gif`.
Height	The height (in pixels) of the image.
Width	The width (in pixels) of the bitmap.

Configuring Buttons to Display Images

You can associate bitmap objects with button controls to display images instead of text, much like a Toolbar icon. Unlike a Toolbar icon, however, a bitmap button control is a command button in the applet. A good example of the use of a Bitmap object with a button control in an applet is the More/Less button. The More/Less button appears in the upper-right corner of many applets. The control uses a bitmap object called BTTNS_MORE that belongs to the Bitmap Category HTML Control Icons.

To configure buttons to display images

1 Define a bitmap object.

See "Creating Bitmap Categories and Bitmap Objects" on page 476.

2 Define the following properties of the button control:

- HTML Bitmap. Defines the Bitmap object used when the button is active.

- HTML Disable Bitmap. Defines the Bitmap object to use when the button is inactive.

For more information about controls, see "About Applet Controls and List Columns" on page 238.

Displaying Images for Field Values

The *Icon Map* object type allows you to render control or list column field values as icons. Each Icon Map is a collection of child objects called Icons. Icon objects are associated with a Bitmap object, which defines the image for the Icon, and corresponds to a particular field value. Controls and list columns have an attribute called Icon Map that allows you to define the icon map object that you want to use for rendering the field values.

The following procedure uses an example to show you how to configure icons for use as field values. The example uses the Status list column on the Activity List Applet. Suppose that the Status field can have values Not Started, In Progress, and Done. You want to configure the Status field to display an icon for each of these values.

NOTE: If you want to use custom icons in a list applet, you must size them in accordance with the list applet's row font size. For example, when using an 8-pt font (standard for Siebel eBusiness Applications), icons should be 23 pixels in width x 14 pixels in height. If you change the list applet row font size dynamically or place an icon larger than 23 x 14 in a row, the list applet rows will be scrambled.

To render fields using Icon Maps

1. Create a Bitmap Category.

   For example, create a Bitmap Category called Activity Status Icons.

   See "Creating Bitmap Categories and Bitmap Objects" on page 476.

2. Create Bitmaps (child object of Bitmap Category) for each image that you want to display and specify the file name of the image.

   For example, create the following records:

Name	File Name
Not Started	notstarted.gif
In Progress	inprogress.gif
Done	done.gif

   See "Creating Bitmap Categories and Bitmap Objects" on page 476.

3. Create a new Icon Map object.

   For example, create an Icon Map named Activity Status.

4. Create one Icon object (child of Icon Map) for each field value and set the following properties:

   - **Name.** Set to the name of the field value.

- **Bitmap Category.** Set to the Bitmap Category you want to show for the field value.
- **Bitmap.** Set to the bitmap object you want to show for the field value.

For example, create the following records:

Name	Bitmap Category	Bitmap
Not Started	Activity Status Icon	Not Started
In Progress	Activity Status Icon	In Progress
Done	Activity Status Icon	Done

5 Set the HTML Icon Map attribute of the list column or control to the Icon Map defined in Step 3.

For example, set the Status list column of the Activity List Applet to the icon map *Activity Status*.

After compiling the changes, the Siebel Web Engine will render the image corresponding to the bitmap when the field value matches one of the icons defined. If the field value does not match any of the icons, the Siebel Web Engine renders the field value itself.

NOTE: You can create an icon named Default in an Icon Map object. If the field value does not match any of the icons, then the Default icon is used for the field. This feature is useful to create an icon to be used with fields that could contain different values, such as URLs. In this case you would set the HTML Type property of the field to be URL and its IconMap property to an IconMap object that contains only one icon named Default.

Defining Images Used in Hierarchical Objects

An *HTML Hierarchy bitmap* is a top-level object that defines the images used by hierarchical objects such as a Tree applet when it is rendered in the user interface. For example, the folders, the plus symbol, and the minus symbol in Figure 123 are icons defined in the Hierarchy Bitmap object. An example of a Tree Applet in the repository is an Account Tree Applet. An example of an hierarchical list applet is Quote Item List Applet.

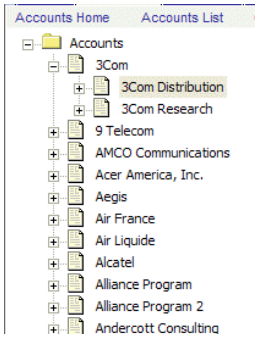


Figure 123. Tree Portion of the Account Tree Applet

Hierarchy Bitmap objects have the following important properties:

- **Name.** The name for the HTML Hierarchy Bitmap object.
- **Collapse Bitmap, Collapse Elbow Bitmap, Collapse Tee Bitmap.** Icons to be used to collapse a node.
- **Expand Bitmap, Expand Elbow Bitmap, Expand Tee Bitmap.** Icons to be used to expand a node.
- **Elbow Bitmap, Tee Bitmap.** Icons to be used for creating an elbow (L) or a Tee (T).
- **Bar Bitmap.** Icon for creating a vertical line.
- **Space Bitmap.** Icon for indents.
- **Open Bitmap.** Icon to be used for a node that is an expanded state.
- **Close Bitmap.** Icon to be used for a node that is in a collapsed state.
- **Leaf Bitmap.** Icon for a leaf node.
- **Arrow Down Bitmap, Arrow Up Bitmap.** Icons for scrolling a tree up or down.

The bitmap objects for these attributes used by the standard tree applets are defined in the Bitmap Category HTML Hierarchy Icons.

The Tree and List objects in tools have an attribute called HTML Hierarchy Bitmap. This attribute can be set to the name of any HTML Hierarchy Bitmap object. This allows the various instances of the Tree object and List object to share these bitmaps.

The Tree Node object has the attributes HTML Open Bitmap and HTML Close Bitmap. These attributes are optional. If you do not specify, the Open Bitmap and Close Bitmap attributes of the HTML Hierarchy Bitmap object will be used. If the attributes are specified, then for that node the specified attributes will be used. This is useful if you want different nodes to have different icons.

For more information about tree applets, see "About Tree Applets" on page 447.

Using Images as Links in Controls

You can use an image as a link with a control that invokes a method by using two attributes of controls called "HTML Bitmap" and "HTML Disabled Bitmap."

The "HTML Bitmap" attribute is used to set the name of the bitmap to be rendered when the control is in an enabled state (that is, the method can be invoked), and the "HTML Disabled Bitmap" is used to set the name of the bitmap to be rendered when the control is in a disabled state.

SWE will use the "HTML Bitmap"/"HTML Disabled Bitmap" attributes only when the "HTML Type" of the control/list column is set to "Link." If set to "Button," the caption property of the control is used as the button label. You can use the "HTML Bitmap"/"HTML Disabled Bitmap" properties with custom HTML types. If you use the tag `<swe:this property="Data" type="Link"/>` within the definition of the custom HTML type in the SWF file, then SWE uses the bitmaps.

If the "HTML Bitmap"/"HTML Disabled Bitmap" attributes are not set, SWE will default to using the "Caption" property for the link.

These bitmaps have to be created in Tools under the Bitmap Category “HTML Control Icons.”

23 Overview of Web Templates and Siebel Tags

Topics in This Chapter

About Siebel Templates

A Web template is a file that contains markup tags (HTML, WML, XML, and so on) interspersed with Siebel tags (prefixed by "swe"). Siebel Web templates define the layout and formatting of the user interface (such as views, applets, and controls). Web browsers require HTML to define the layout and formatting of a page. Siebel Web templates provide this HTML layout information to the Siebel Web Engine when rendering Siebel objects in the repository definition of the application. Wireless applications are rendered in the same manner except for the fact that the markup language in the templates is WML or XML. This section focuses on the configuration of Web (HTML) applications, but many of the concepts are generic across markup languages.

Templates are filled with data and user interface elements by associating views, applets, controls, and other objects defined in Siebel Tools with them. Each view, applet, or control, is mapped to a placeholder in the template. For example, you may have a View object with three applets. You associate a View Template with the view, and map each applet to a placeholder in that template.

An important feature of Siebel Web Templates is that they can be shared between many objects in the repository. Because a template has only placeholders, any number of repository objects can be mapped to a specific placeholder. This allows you to propagate style or structural changes to numerous user interface elements by changing only one template. A typical Web application will contain on the order of 5-50 templates, which together form the bases for several hundred views and applets. For instance, a template which defines the layout and formatting of a standard list applet can be shared among all list applets repository definitions in an application.

The reusability of templates is further enhanced in that the Siebel Web Engine skips over template placeholders which are not mapped in the repository. If a placeholder is not mapped, then it and the HTML contained in between the Siebel tags that define the placeholder are simply ignored. Thus, if the template contains layout for a 10-column wide list applet, but only 2 of the columns are mapped, the other 8 are simply ignored.

Siebel applications provide numerous applet and view templates with the product, which are extremely flexible; you may not have to modify any of the applet and view templates to support your migrated application. However, in some cases (especially customer and partner applications) you may wish to modify the default templates to reflect your corporate look and feel, or, in some cases, create an entirely new template. Siebel templates must use valid HTML. Adding JavaScript beyond what is already generated by the Siebel Web Engine is not recommended. If it is necessary to add JavaScript, it should be done in Siebel Tools using Browser Script.

NOTE: You can view Web templates using Siebel Tools, but you modify templates using an external editor. For more information about using Siebel Tools to view Web templates, see *Using Siebel Tools*.

To allow for even greater flexibility, Siebel eBusiness Applications have provided a mechanism in which a particular template file can include another one. This device is used, for example, to separate handling of the title of an applet from the body. A standard applet layout can be defined once and combined with multiple different title layouts by including a template file that defines the title within the applet template.

By convention, the filenames of Siebel Templates take the .SWT extension; for example, CCPageContainer.SWT, CCHomePageView.SWT, and so on. This Siebel-suggested convention is an abbreviation for Siebel Web Template.

You do not have to follow this convention; the Siebel Web Engine recognizes and interprets the files correctly regardless of how you name them. However, ending your Siebel filenames with .SWT may help you.

NOTE: Template files are typically stored in the Web Template directory under your Tools installation directory. The Filename property references the Web Template object type.

The layout and style of HTML Web pages is dynamic, which allows simultaneous support for multiple browser types and versions. Siebel Web templates support conditional branching. Conditions are evaluated based on the results of a business service.

How SWE Generates HTML Files

After you configure your application and deploy it on your Web site, it becomes available for viewing through a client's browser.

When a client requests a specific view (either through the application URL directly or by clicking the appropriate link from within another page), the Siebel Web Engine does the following:

1. Retrieves the object definition of the view from the .srf and retrieves the object definition of each applet in that view.
2. Retrieves the data specified in the object definition from the data manager layer of the Application Object Manager.
3. Matches this data with the template specified by the view and each applet within it.
4. Renders this view by using the placeholders in the template to define where each element (control, list element) in the object definition is to be placed and how it should be formatted.

When the user views the generated HTML file in a Web browser, it is rendered as a Web Page, and includes all the layout specified in the original template as well as the data and controls retrieved.

Figure 124 shows how Siebel Web Engine generates HTML output using templates, repository definitions, and HTML.

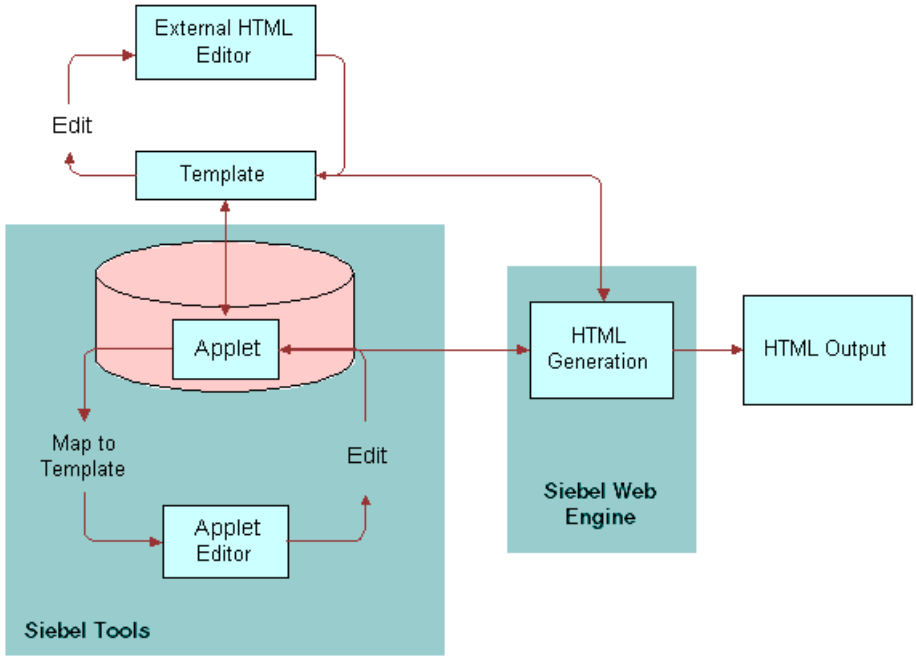


Figure 124. How Siebel Web Engine Generates HTML Output

Types of Templates

The templates fall into one of several groups, depending on the purpose of the template or what the template contains.

- **Web Page template.** Specifies the layout of the whole display. Has information about where the screen bar/view bar/view should appear.
- **Page Container template.** Used as container pages for view templates. The overall purpose of the page container is to provide a structure for the overall application. There is one page container per application, but views can be flagged indicating they should not use the container page (for example, the login page cannot use the page container).
- **View template.** Used for displaying a view; specifies where to lay out applets and other page-level controls on the view, and what the formatting of the view should be.
- **Applet template.** Specifies where to lay out fields and controls for an applet. Also specifies the formatting for elements within the applet.

  Applets can have more than one mode. Each mode is associated with a template. The types of modes are:

  - **Base:** Read-only mode for displaying but not editing data. Views appear by default in Base mode.
  - **Edit.** Mode for editing an existing record.

    If New/Query templates do not exist, Edit is used when creating and querying.
  - **Edit List.** Allows users to edit fields in a list applet.

    Edit List is used for editing, creating, and querying in employee applications running in high interactivity. Standard interactivity applications, such as customer and partner applications, do not use Edit List, so you must define an Edit mode template (even if Edit/New/Query templates are defined).
  - **New:** Mode for creating a new record.
  - **Query:** Mode that allows you to perform a query-by-example (QBE).

NOTE: New and Query should only be used if they are different from Edit. Otherwise, Edit is used.

- **Formatting templates.** Templates that allow you to create custom HTML types, such as specialized controls, list items, and page items. These templates have the extension .SWF (Siebel Web Format). For more information about .SWF files, refer to "Creating Custom HTML Control Types" on page 541.

Your application can contain other pages, of course, that do not contain any Siebel tags. For example, you may have an About This Application help page. However, this page, by definition, is not a template.

About Web Page Templates

The application is associated with a set of templates through properties in the Application object definition. These properties include Container Web Page, Error Web Page, Login Web Page, Logoff Acknowledgement Web Page, Sort Web Page, and Acknowledgement Web Page. Each property identifies a template to use in a given circumstance.

- **Acknowledgement Web Page.** The Web page displayed after the user logs in. This page is used as the first page the user is taken to after a successful login, except in the case of a login after a time-out. In the case of a login after a time-out, the user is taken to the view to which he or she was trying to navigate when the time-out occurred.
- **Acknowledgement Web View.** The Siebel view displayed after login. This page is used as the first view the user is taken to after a successful login, except when:
  - A user logs in after a time-out. In this case, the user is taken to the view to which he or she was trying to navigate when the time-out occurred.
  - Explicit login is specified for an SI mode view. In a standard interactive application such as eService, if view accessible through the home page as the Explicit Login property is set to TRUE (anonymous browsing), then after successfully entering the login credentials, the user is taken to this view instead of the Acknowledgement Web View defined in this property.
- **Container Web Page.** A page that defines the structure of the application. This page can contain the common UI components like screen bars, view bars, logos, and so on. This page can be used to define the HTML Frame definition document for the application. All views and pages (optionally) are shown within the context of the container page. "About the Container Page" on page 487.
- **Error Web Page.** The page to use when an error occurs in the application.
- **Login Web Page.** The page to use as the Login page.
- **Logoff Acknowledgement Web Page.** The page to which the user is taken after logging off the application.
- **Sort Web Page.** The page to be used to create a dialog to perform an advanced sort of list applet columns.

About the Container Page

The container page is the outermost template; it references view templates that in turn reference applet templates. The Web Page-Layout (Container Page) contains markup language and SWE tag elements that define the Web equivalent of the application window. You can see this template's logic in CCPageContainer.swt. The container page template, like view and applet templates, is processed by the Siebel Web Engine.

Container Page Areas

In the Web Page Layout Container Page, you find the following elements:

- The top of the container page contains markup such as corporate banner, as well as Siebel tags for predefined queries (favorites).
- The screen (tab) bar is generated beneath these as a table, and loaded by means of the SWE logic associated with the `<swe:screenbar>`, `<swe:screenlink>`, and `<swe:now-control>` tags.
- The view bar is also loaded, into the left-side portion of the page, by means of the `<swe:viewbar>`, `<swe:viewlink>`,and `<swe:now-control>` tags.

Once the container page is loaded, with screen and view names displayed, the screen and view names function as hyperlinks.

- When a screen name is clicked, the template for the default view for that screen is obtained, and the view is generated and displayed.
- When a view name is clicked in the view bar, the view template that is referenced in the view's object definition is loaded.

The Siebel Web Engine processes the set of tags in the view template to incorporate applets into the page. The view object definition identifies the applets to appear in specific sectors, and the templates for these are obtained. Similarly, tag references to controls in each applet are resolved by obtaining the corresponding controls from the repository, which are loaded into the Web page as specified in the applet's template. The container page can contain frames to support independent updating and scrolling of the various areas of a page. The use of frames is described in the next section.

About HTML Frames in the Container Page

HTML frames are available to use in the application's container page and in View templates.

Various frames are used in the container page of an application to provide independent updating and scrolling of areas such as toolbars, menus, and the main content area. Figure 125 shows multiple frames in a container page.

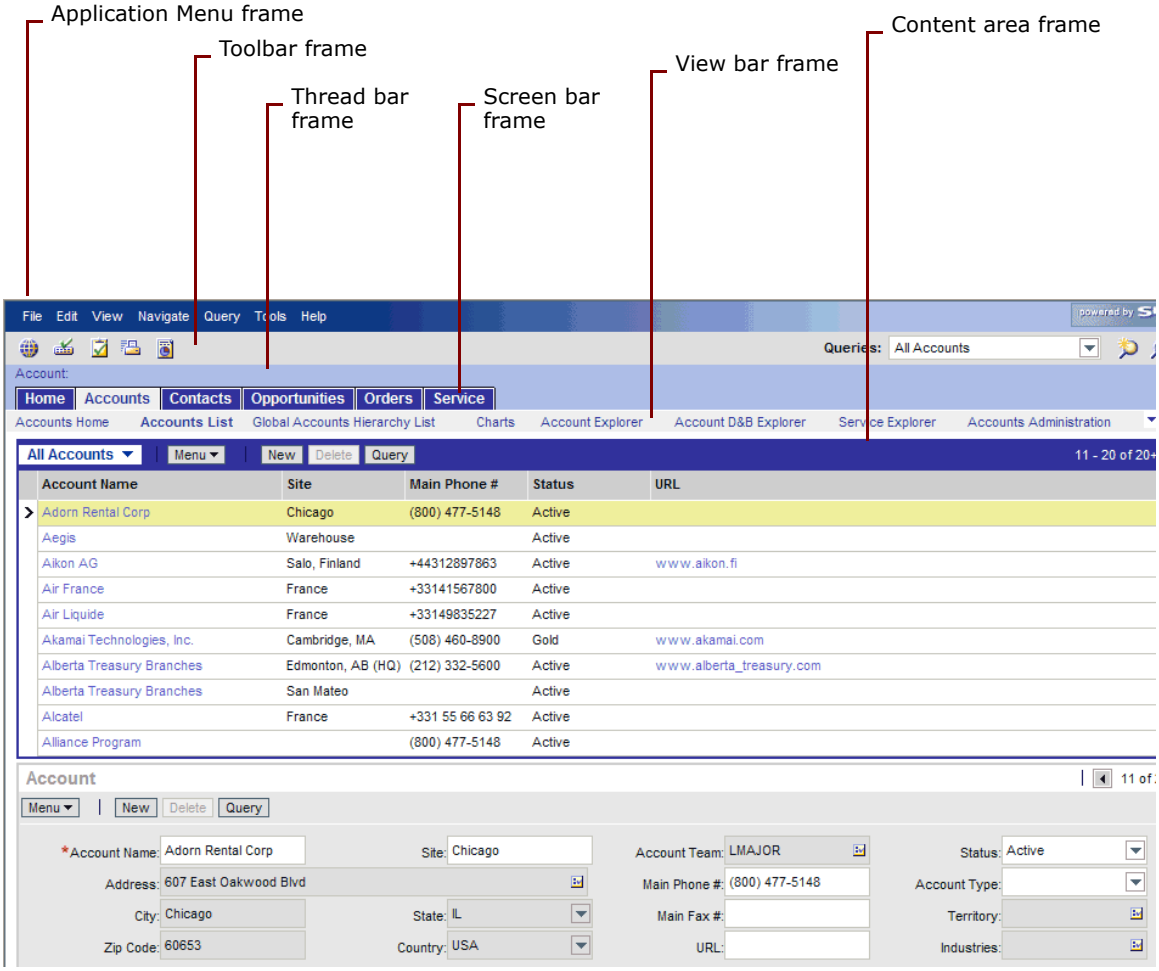


Figure 125. HTML Frame

In a View template, applets can be grouped into separate frames, although this is considered a non-standard practice except in cases where independent refresh or independent scrolling is a significant requirement.

Rather than using the HTML <frame> and <frameset> tags, the <swe:frame> and <swe:frameset> tags are used in the Siebel applications so that SWE is aware of the frame names, and can control refresh and the targeting of URLs. These two SWE tags are described as follows:

- <swe:frameset>

  Purpose: This tag is analogous to the HTML frameset tag and is used to define the set of frames contained in the document. This tag is rendered by SWE as an HTML <frameset> tag. The body of this tag can only contain the <swe:frame> tags described below.

  Usage: <swe:frameset htmlAttr="xxx"> ... </swe:frameset>

  Attributes:

  - **htmlAttr.** This attribute can be used to specify the attributes for the HTML <frameset> tag. For example, htmlAttr="rows='89,25,*'" will support a layout in which the frames that belong to the frameset will take up 89 pixels, 29 pixels, and the rest of the window respectively.

- <swe:frame>

  Purpose: This tag is used to mark the beginning and end of the contents to be placed into a frame. SWE renders this tag as an HTML <frame> tag, with its src attribute set to a SWE URL that will retrieve the contents of the frame. This tag should be placed within the body of the <swe:frameset> tag.

  Usage: <swe:frame type="xxx" name="yyy"> .... </swe:frame>

  Attributes:

  - **Type.** The type attribute is used to indicate the nature of the contents of the frame. SWE uses this information to decide when to refresh this frame. SWE supports the following values for this attribute.
    - **Toolbar.** In a container page template, specifies that the frame contains the toolbar.
    - **Screenbar.** In a container page template, specifies that the frame contains the primary tab bar.
    - **Viewbar.** In a container page template, specifies links to views and categories of views.
    - **View.** In a container page template, specifies that the frame contains the current view, that is, the content area.
    - **Page.** In a container page template, specifies that the frame contains a Web page. These frames will not be refreshed after initially loading.
    - **Applet.** In a View template, specifies that the frame contains an applet.
    - **Content.** Supports multiple views on a page. The Content type frame defines the content area. It will contain a frame of type View that shows the main view. It can also contain one or more frames of type AltView to show alternate views, like the search center.
    - **AltView.** Used to designate subframes to show one or more alternate views in the content frame, such as the search center, in addition to the one in the View frame.
    - **Name.** This attribute can be used only when the type of the frame is Page. In this case, you can use this attribute to specify a name for the frame. For other frame types, SWE will generate standard names for the frames.

  NOTE: SWE supports nested framesets. In this case the <swe:frame> tag will contain a <swe:frameset> tag, and the Type attribute of the outer <swe:frame> tag is set to Page.

HTML Frames in Container Page Templates

A container page template is used to create the frame definition document for the application. Note the following implementation details of <swe:frame> and <swe:frameset> tags in container pages:

- You do not have to define the contents of a frame using the <swe:include> tag although it is recommended. The contents can be placed directly into the body of the <swe:frame> tag.
- The contents of the <swe:frame> have to be complete HTML documents, that is, they should contain the HTML document structure tags like <html>, <head>, <body>, and so on. This includes the view templates as well.
- The contents of the <swe:frame> tag when the type is View should contain only the <swe:current-view/> tag.

The following <swe:frameset> definition is from the standard container page, CCPageContainer.swt:

```
<swe:frameset htmlAttr="rows='60,21,25,*' border='0' frameborder='No'">

   <swe:frame type="Page" htmlAttr="marginheight='0' marginwidth='0'noresize
   scrolling='No'">

      <swe:include file="CCFrameBanner.swt"/>

   </swe:frame>

   <swe:frame type="Screenbar" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='No'">

      <swe:include file="CCFrameScreenbar.swt"/>

   </swe:frame>

   <swe:frame type="Viewbar" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='No'">

      <swe:include file="CCFrameViewbar.swt"/>

   </swe:frame>

   <swe:frame type="View" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='Auto'">

      <swe:current-view/>

   </swe:frame>

</swe:frameset>
```

Support for Multiple Views on a Page

The SWE framework supports showing multiple views simultaneously on a page. The multiple views consist of a Main view and one or more Alternate views. The main view is the view that is selected using the view bar (level two or three) for a given screen. There is only one main view. Alternate views are other views that can be shown along with the main view: for example, the Search View that shows applets that can be used for find/search operations.

The multiple views shown on a page can be placed into separate HTML frames or can share the same frame. Multiple views can also be shown with the main view in the main browser window and a single alternate view in a pop-up window.

In employee applications that use high interactivity, only the main view can be in high interactivity. The alternate views will be shown in standard interactivity. It is recommended that you configure alternate views as simple views without any complex navigation links.

NOTE: The examples given here describe creating multiple view layouts when HTML frames are used. The process is similar when frames are not used. In such cases, HTML tables can be used in the place of frames and framesets to position the views.

To support multiple views, the structure of framesets and frames used in the application has to be modified. Defined framesets and frames in the application's container template and in the view template were discussed earlier in this chapter. In addition, there is another layer, the Content Container (the container page for the Content area).

The frame of type View which was in the Application's Container page should be replaced with a frame of type Content. This frame defines the area where one or more views can be loaded. Initially this frame will contain a frameset that will have the View type frame.

The structure of the container template is given in the example below:

```
<swe:frameset htmlAttr="rows='80,50,50,*' border='0' frameborder='No'">

   <swe:frame type="Page" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='No'">

      <swe:include file="CCBanner.swt"/>

   </swe:frame>

   <swe:frame type="Screenbar" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='No'">

      <swe:include file="CCScreenbar.swt"/>

   </swe:frame>

   <swe:frame type="Viewbar" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='No'">

      <swe:include file="CCViewbar.swt"/>

   </swe:frame>

   <swe:frame type="Content" htmlAttr="marginheight='0' marginwidth='0' noresize
   scrolling='Yes'">

      <swe:include file="CCMainView.swt"/>

   </swe:frame>

</swe:frameset>
```

The file CCMainView.swt defines a frameset that contains the main view.

```
<swe:frameset htmlAttr="cols='100%' border='0' frameborder='No'">
    <swe:frame type="View" htmlAttr=" noresize scrolling='Yes'">
        <swe:current-view/>
    </swe:frame>
</swe:frameset>
```

After making this change, the application should behave as before. All that was changed was the introduction of one additional layering of frames in the content area. The previous application container page template that had the View frame without the outer Content frame does not generate any errors, but does not allow showing multiple views in the application. All the application container templates should be modified to use the Content frame.

To show additional views in the content area, a different Content Container page in the Content frame should be loaded. This can be done by invoking the method LoadContentContainer from a control or page item. The Content Container to be loaded should be passed in using the User Property Container.

NOTE: This should be set to the Web Template Name of the content container page and not to the .SWT file name. For example, to show the search view along with the main view, create a content container page (for example, CCSMainAndSearchView.swt), and load it using the LoadContentContainer method. CCSMainAndSearchView.swt contains the tags to load the main view and search view into two frames as shown:

```
<swe:frameset htmlAttr="cols='100%' border='0' frameborder='No'">
    <swe:frame type="View" htmlAttr="noresize scrolling='Yes'">
        <swe:current-view/>
    </swe:frame>
    <swe:frame type="AltView" name="Search" htmlAttr="noresize scrolling='Yes'">
        <swe:view name="Search View" id="Search" />
    </swe:frame>
</swe:frameset>
```

The main view is still called the <swe:current-view> tag. Alternate views are referred to using the <swe:view> tag.

- <swe:view>

  Syntax:

  ```
  <swe:view name="xxx" id="yyy">
  ```

  Attributes:

  - **Name.** Name of the Alternate View

- **Id.** An Id for the location (or zone) occupied by this view. This Id will be used to replace this view with another view in its place.

The <swe:frame> tag contains alternate views called AltView.

To switch from showing the Search and Main views to showing only the Main View, invoke the LoadContentContainer method again, this time passing in the CCMainView.swt based container page.

About View Templates

A view is a collection of applets displayed on a screen at the same time. It consists of a single window displaying related data forms and lists (applets). The user can select the current (active) view from either the Screenbar (the default view for that screen), the second-level Visibility picklist, a third-level tab, the fourth level Category-view picklist, the thread bar, the history list, history forward and back buttons, or by a drilldown from another view. Access to particular views is determined by the four navigational constructs in the physical UI.

View Templates are associated with a view through the View Web Template object definition. A view template primarily contains placeholders for applets as specified by the <swe:applet> tag. The mapping of specific applets to these placeholders is done visually through the View Layout Editor.

The following is an example of a view template.

```
<!-- Template Start: CCViewBasic.swt -->

<!------------ Page Title ------------->

<title>

<swe:this property="Title"/>

</title>

<!----------- Salutation applet and Search Applet, table 3.1 ------>

<table border="0" cellspacing="0" cellpadding="1" width="100%">

   <tr>

      <td width="66%"><swe:applet id="101"/> </td>

      <td width="33%"><swe:applet id="201"/> </td>

   </tr>

</table>

<!-------- End Salutation applet and Search Applet, table 3.1 ------>

<!------------ Regular Applet(s) ---->

<swe:for-each count=5 iteratorName="currentId" startValue="1">

   <swe:applet id="swe:currentId"/>
```

```
</swe:for-each>

<!------------ Special Applet(s) ---->

<swe:for-each count=3 iteratorName="currentId" startValue="11">

   <swe:applet id="swe:currentId"/>

</swe:for-each>

<!-- Template End: CCViewBasic.swt -->
```

Notice that each <swe:applet id=x> tag acts as a placeholder for an applet's location in the view template. This same view template can be used to show different views by mapping the view's applets to these placeholders. In the default view templates shipped with Siebel applications, swe:applet tags with IDs of 101 and 201 are used to show the salutation and search applets at the top of the views. The IDs 1 through 10 are used to show the main applets in the view, and the IDs starting with 11 are used to show some special applets that appear at the bottom of some views.

About HTML Frames in View Templates

HTML frames can be used in View templates to create a frame definition document to show the Applets in the View. SWE will refresh these frames only when one or more of the Applets contained in a frame has new data.

NOTE: You can use frames in a View template only if frames are also used in the container page and there is a separate frame in the container page for the View.

In a View template, applets can be grouped into separate frames, although this is considered a non-standard practice except in cases where independent refresh or independent scrolling is a significant requirement. One situation where frames are required in the content area is when displaying an explorer View, in which a tree applet occupies a frame on the left and the corresponding list applet occupies the frame on the right. Another situation requiring frames is when the user activates a search, at which time a Search frame, and subsequently a Results frame, are activated in the right portion of the content area.

The following shows the implementation details of frameset definitions in View templates:

- When placing Applets into frames you need to make sure that at least one <swe:applet> tag within a frame gets mapped to an Applet in the repository. Otherwise, empty frames will occur.
- When a <swe:frame> block contains a <swe:applet> tag, its type attribute should be set to Applet.

Given below is an example of a view template that uses frames:

```
<!-- CCView_33_66_Frame.swt start -->

<swe:frameset htmlAttr="cols='33%,66%'' border='1' frameborder='Yes'">

<!-- Column 1 Applets -->

<swe:frame type="Applet" htmlAttr="marginheight='0' margin width='0' scrolling='Auto'">

<swe:for-each count=10 iteratorName="currentId" startValue="101">
```

```
    <swe:applet id="swe:currentId" hintText="Applet" var="Parent">
        <!--start applet-->
            <swe:this property="FormattedHtml"/>
        <!--end applet-->
        </swe:applet>
</swe:for-each>
</swe:frame>
<!-- Column 2 Applets -->
<swe:frame type="Applet" htmlAttr="marginheight='0' marginwidth='0' scrolling='Auto'">
<swe:for-each count=10 iteratorName="currentId" startValue="201">
    <swe:applet id="swe:currentId" hintText="Applet" var="Parent">
        <!--start applet-->
            <swe:this property="FormattedHtml"/>
        <!--end applet-->
        </swe:applet>
</swe:for-each>
</swe:frame>
</swe:frameset>
<!-- CCView_33_66_Frame.swt end --> </HTML>
```

About Applet Templates

The Applet Web Template child object type (of Applet) makes it possible to specify multiple templates for a single applet, each template file associated with one or more modes. The Applet Web Template object type has the following important properties:

- **Type.** Indicates the edit mode that the applet template supports (such as Edit or New).
- **Web Template**. Provides the name of the Web Template used for that mode.

The Applet Web Template Item child object type (of Applet Web Template) defines the mappings between controls and list columns to placeholders in the Web template file. The Applet Web Template Item object type has the following properties:

- **Name.** Name of the object definition, generally the same as the Control property.
- **Control.** Specifies the name of the control as it is to appear.

- **Item Identifier.** This specifies a unique numeric identifier for each control, generated in the layout editor. The value is used in the markup language tag that specifies the corresponding control in a template, binding the control to a specific position on the page.
- **Type.** Consists of the value Control, List Item, or Web Control, indicating what kind of control the applet Web template item defines.

About Form Applet Templates (Grid-Based)

Grid Layout applet Web templates, Siebel tags, and enhanced features in the Web Layout Editor allow you to modify form layout without having to directly modify the underlying applet Web templates.

Standard applet Web templates (not grid-based) use placeholder tags to define an applet's layout. You could use the Web Layout Editor in Siebel Tools to map controls to any of the available placeholders, but you cannot use Web Layout Editor to change the layout of the placeholders themselves. To change the layout of the placeholders in these templates, you have to directly modify the applet Web template file.

Grid Layout applet Web templates use a pair of Siebel tags (`<swe:form-applet-layout>` and `</swe:form-applet-layout>`) that do not use placeholder tags. Instead they serve as a single container for all controls in the main body of a form applet. These tags allow you to use the Web Layout Editor as a WYSIWYG (what you see is what you get) design tool to configure the layout of form applets. In fact, this is the only way you can configure the layout of an applet based on a grid-based applet Web template.

The Grid Layout Web templates are:

- **CCAppletFormGridLayout.** This template may be used for form applets. See Figure 126.

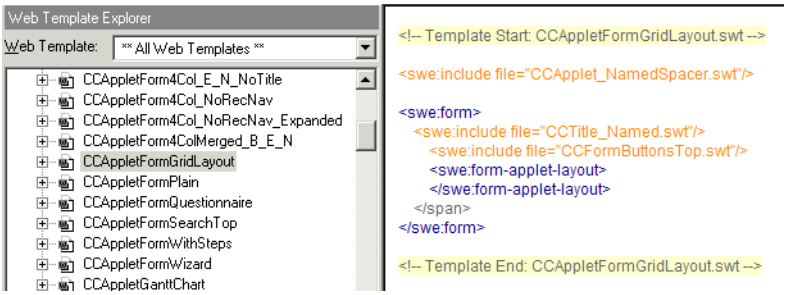


Figure 126. Grid-Based Applet Web Template for Form Applets

- **CCAppletPopupFormGridLayout.** This template may be used for popup form applets. See Figure 127.

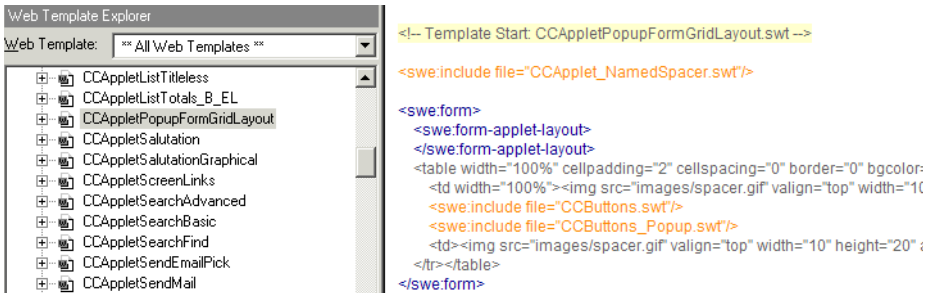


Figure 127. Grid-Based Applet Web Template for Popup Form Applets

Grid Layout applet Web templates consist of a body region and a header and footer region. The body region of the template is defined by the `swe:form-applet-layout` tag and contains no placeholder tags. However, the header and footer regions do use placeholder tags for buttons such as New and Save. You cannot edit the layout of header and footer regions using the grid layout features of the Web Layout editor.

The following list summarizes how grid-based applet Web templates differ from standard (not grid-based) applet Web templates:

- With grid-based templates, you can modify the layout of the form using Siebel Tools without having to modify the Web template itself.
- With grid-based templates, Labels and Controls behave as separate items in the Web Layout Editor. This allows you to place them independently in the applet layout. However, Labels and Controls are really a single object in the repository with one set of shared properties.
- Grid-based templates do not automatically compress empty space in a column.

You can modify the background colors of applets based on Grid Layout Web templates by modifying the appropriate selectors in the cascading style sheet, main.css.

- When the applet is the parent applet (the top applet on a view) modify the .AppletStyle1 selector as shown in the example below:

```
/*Parent Applet Style*/

.AppletStyle1{background-color : #f00000; color:#00f0ff; }
```

- When the applet is the child applet (not the top applet on the view) modify .AppletStyle3 selector as shown in the example below:

```
/*Child Applet Style*/

.AppletStyle3 {background-color : #f0f000; }
```

For more information about main.css, see *Developing and Deploying Siebel eBusiness Applications*.

Grid-based templates make configuring the layout of form applets much simpler than configuring the layout of applets based on standard (not grid-based) templates. Using Grid Layout templates makes the process of designing a form applet similar to working in other palette-based graphics design applications.

See "About Grid Layout" on page 266 for more information about Grid Layout.

About Form Applet Templates (Non Grid-Based)

A Form Applet can appear in any of the four major modes—Base, Edit, New, and Query. The following is an example of a Form Applet template for use in Edit, New, and Query modes. Applets to be used in Base mode are similar except that they do not contain the **<swe:form>** tag.

```
<swe:control id="1100">
    <div class=CmdTxt>
        <swe:this property="FormattedHtml" hintText="Outside Applet Help Text"/>
    </div>
</swe:control>
<table class="AppletStyle1" width="100%" align="center">
    <swe:form>
        <tr>
            <td colspan="2">
            <swe:include file="CCTitle.swt"/>
            </td>
        </tr>
        <tr>
            <td>
            <swe:error>
            <swe:this property="FormattedHtml"/>
            </swe:error>
            </td>
        </tr>
    <swe:for-each startValue="1301" count="10" iteratorName="currentId">
    <swe:control id="swe:currentId" hintMapType="FormItem">
```

```
        <tr valign="top">

            <td class="scLabelRight"> 

            <swe:this property="RequiredIndicator"
            hintText="Required"/>

            <swe:this property="DisplayName" hintText="Label"/>

            </td>

            <td class="scField">

            <swe:this property="FormattedHtml" hintText="Field"/> 

            </td>

        </tr>

    </swe:control>

</swe:for-each>

</swe:form>

</table>
```

The main tags that appear in this template are <swe:form>, <swe:control>, and <swe:error>.

- <swe:form>

  The <swe:form> tag is analogous to an HTML <form> tag and encloses a section of a page that accepts user input. The main attributes of this tag are htmlAttr and Name, both of which are optional.

  The values of the htmlAttr should be valid attributes of the HTML <form> tag other than method, name, and action. These attributes will be used as is with the HTML <form> tag that is generated. The name attribute creates an HTML form with the specified name. If this attribute is not specified, an internally generated name is used.

- <swe:control>

  The <swe:control> tag specifies placeholders for controls. The main attributes of this tag are id, which maps the control to the placeholder, and the property, which specifies the property of the control to be rendered. The values for the property attribute that are germane to form applets include FormattedHTML, DisplayName, and RequiredIndicator.

  The FormattedHTML property causes the data value of the control to be rendered, while the DisplayName corresponds to the Caption property. The RequiredIndicator results in specific HTML being rendered if the underlying Business Component Field is required.

- <swe:error>

  When a server side error occurs on submitting a form, the same page will be shown again with the error message displayed within the page. The <swe:error> tag denotes the location of this error message. The only attribute of the tag is a property whose value must be FormattedHtml. This results in the contents of the error message to be displayed. If when the form is rendered there are no errors, the contents of the <swe:error> tag are skipped.

  NOTE: For errors that occur outside of a form submission, the application's Error Page will be used.

In many cases, you can forego read-only forms (Base mode) and use persistently editable forms because task activities are often data editing and input. This type of form improves usability because users can enter data without first having to click an edit button and then wait for the form to appear in edit mode.

If an applet is set to be in the Edit mode in a view (as specified by the mode property of the View Web Template Item), this applet is never shown in the Base mode. If you update the field values in this applet and commit the change, the applet continues to be shown in this mode after the changes are written to the database. You can, however, invoke a method like NewQuery or NewRecord on an applet that is shown in an Edit mode to show it in the Query or New modes. After executing the query or writing the new record, the applet is shown in the Edit mode.

About List Applet Templates

Standard list applets allow simultaneous display of data from multiple records. The standard list displays data fields in a multi-column layout with each record of data represented in a row. In addition to textual data, lists also support images in JPEG and GIF formats and edit controls such as check boxes, drop-down lists, MVGs, and text fields.

A single row at a time within the standard list applet is selected for editing by clicking in the far left column selection area. When selected, the fields within the row can activate either input or edit controls. Clicking the New button creates a new row with a series of blank fields for the user to populate.

Persistently Editable List Applets

Edit List mode renders list applets as persistently editable. The purpose of an editable list applet is to allow users to modify the records in a list applet without switching to an edit page.

The editable list applet has the following features:

- Editable cells displayed as text input, list box, or mini-buttons
- Modified records that can be saved individually

A given list applet exhibits different behavior and appears differently depending on whether it is part of a view being displayed using standard or high interactivity:

- In standard interactivity, there is a row selector control that allows the user to select a specific row for editing. When selected, the fields within the row can activate either input or edit controls. Clicking the New button creates a new row with a series of blank fields for the user to populate.

- Row selection using high interactivity is done by clicking on any area within a row in the list. Therefore, the row selection is redundant, and the control is automatically deleted when the list is rendered. Also, because high interactivity uses an implicit save model, a Save control is not required. When a user steps off the current record, the changes are automatically saved.

Sample List Applet Template

```
<table width="100%" cellspacing="0" cellpadding="0" border="0" align="center">
<swe:form>
   ...
<swe:list>
<!-- List Header Section Start>
   <swe:control id="147">
      <td class="Header" align="center">
      <swe:this property="DisplayName"/>
      </td>
   </swe:control>
   <swe:select-row>
      <td width="42" align="center" class="Header"> 
      </td>
   </swe:select-row>
<swe:for-each startValue="501" count="20" iteratorName="currentId">
   <swe:control id="swe:currentId">
      <td align="swe:this.TextAlignment" class="Header">
      <swe:this property="ListHeader"/>
      </td>
   </swe:control>
</swe:for-each>
   <swe:control id="142">
      <td class="Header" align="center">
      <swe:this property="DisplayName"/>
      </td>
```

```
    </swe:control>
<!-- List Header Section End>
<!------------ Loop for all 7 records, List Body ------------->
<swe:for-each-row count="7">
<tr class="swe:this.RowStyle">
    <swe:control id="147">
        <td width="42" align="center" class="Row">
        <swe:this property="FormattedHtml" hintMapType="Control"/>
        </td>
    </swe:control>
    <swe:select-row>
        <td width="42" align="center" class="Row">
        <swe:this property="FormattedHtml" />
        </td>
    </swe:select-row>
<!-- ---------- List Field Values (501-520) ------------->
<swe:for-each startValue="501" count="40" iteratorName="currentId">
    <swe:control id="swe:currentId">
        <td align="swe:this.TextAlignment" class="Row">
        <swe:this property="FormattedHtml"
        hintText="Field"/>
        </td>
    </swe:control>
</swe:for-each>
<!-- ---------- Per-record Control Buttons ------------->
    <swe:control id="142">
        <td align="center" class="Row">
        <swe:this property="FormattedHtml" hintMapType="Control"/>
        </td>
    </swe:control>
```

```
</tr>

</swe:for-each-row>

<!-- ---------- End Loop, List Body ------------->

</swe:list>

   ...

</swe:form>

</table>
```

Tags that typically appear in a list applet template include swe:form, swe:list, swe:control, swe:select-row, and swe:for-each-row.

- <swe:form>

  As with form applets, this tag encloses an editable section. It is therefore used for editable list applets.

- <swe:list>

  This tag encloses the section of the template that contains the list header and body. For applications that use high interactivity, the section between the start and end of the swe:list tags is replaced by the specialized List Control that supports capabilities such as resizing columns, and so on. This tag is ignored for standard interactivity applications.

- <swe:control>

  This tag defines a placeholder for List Columns. The property attribute takes the same values as in the case of form applets - DisplayName for the Display Name attribute of the list column object, FormattedHTML for the data value. In addition, certain attributes of a list column object can be used to control the attributes of an HTML element contained within the enclosing swe:control tag. For example, the align attribute of a contained TD tag can be set to be equal to the Text Alignment property of the enclosing list column as follows:

  ```
  <td align="swe:this.TextAlignment">
  ```

- <swe:select-row>

  This tag is used to render check boxes for selecting a row for the purposes of multi-selection. This is described in greater detail in the section on multi-select lists.

- <swe:for-each-row>

  This tag encloses the section of the template that is to be repeated for each list row.

Current Record Selection in List Applets

The Web client has a feature that allows users to select a record as the currently active record in a list applet in the Base and the Edit List modes.

NOTE: This applies only to standard interactivity applications. For high interactivity applications, row selection is achieved by clicking anywhere within the current row.

To select a record as the currently active record

1. Add a control to all list applets that invokes the method PositionOnRow.
2. The HTML Row Sensitive property of this method should be set to TRUE.
3. Place this control on the list applet where you want the link to select the row. You are able to select the record by clicking on this link.

SWE provides two options to show the currently selected record. These options can be used together or individually:

4. You are able to specify the formatting to be used for the currently selected row.

   This feature is based on changing the CSS style sheet class associated with a row (for example in a <TR> tag) to specify different formatting information. You can associate a list applet with a named style to be used for formatting its rows. You can define this attribute in the cfg file used by the application under the SWE section. (This is currently limited to all list applets used by an application so they have the same row formatting style.)

   ```
   ListRowStyle              = "Siebel List"
   ```

   You can specify any name for the row style. The actual style sheet classes used by this named style is specified in a new file type called the Siebel Web Style file (similar to the Siebel Web Format file used for custom HTML types). The Siebel Web Style files (SWS files) will have the extension .sws, and should be installed in the same folder as the template files.

   As in the case of the SWF files, the SWS files used by an application are specified in the .cfg file of the application. There are two parameters that define the SWS file. One defines the file defined by the Siebel application teams and the other the file that can be defined by customers to either override the styles specified by the application teams or to add new styles. The parameters should be defined in the SWE section of the cfg file.

   ```
   SystemSWSName             = CCStyles.sws

   ;UserSWSName               = // for customer use only
   ```

   In the SWS file you can define the style sheet classes to be used with a named style using two new SWE tags and conditional tags like <swe:if> or <swe:switch>. These tags are described as follows:

- `swe:style`

  Usage: `<swe:style type="XXX" name="YYY">`

  Attributes:

  - **type.** Currently supports only one value which is RowStyle. Other values will be supported in the future.

- **name.** Name of the style (like Siebel List)

swe:class

Usage: `<swe:class name="XXX"/>`

Attributes:

name. Name of the CSS style sheet class. The style sheet that defines this class should be loaded through the template.

The following is a sample entry in a SWS file that will achieve the same formatting as the conditional tags that were put in the template file using the earlier approach:

```
<swe:style type="RowStyle" name="Siebel List">
<swe:switch>
   <swe:case condition="Web Engine State Properties, IsErrorRow">
      <swe:class name="listRowError"/>
   </swe:case>
      <swe:case condition="Web Engine State Properties, IsCurrentRow">
   <swe:class name="listRowOn"/>
   </swe:case>
      <swe:case condition="Web Engine State Properties, IsOddRow">
   <swe:class name="listRowOdd"/>
   </swe:case>
      <swe:case condition="Web Engine State Properties, IsEvenRow">
   <swe:class name="listRowEven"/>
   </swe:case>
   <swe:default>
   <swe:class name="listRowOff"/>
   </swe:default>
</swe:switch>
</swe:style>
```

In the template file used by the list applet, the conditional tags used earlier should be replaced with a new property of the applet called "RowStyle" that can be set to the class attribute of any HTML tag. The format used for specifying the RowStyle property of the list applet is similar to that used for specifying the TextAlignment property of a list column.

```
<swe:for-each-row count="7">
```

```
<tr class="swe:this.RowStyle">

<swe:for-each startValue="501" count="20" iteratorName="currentId">

   <swe:control id="swe:currentId">

   <td align="swe:this.TextAlignment" class="Row"><swe:this
   property="FormattedHtml" hintText="Field" hintMapType="ListItem"/></td>

   </swe:control>

</swe:for-each>

</tr>

</swe:for-each-row>
```

5 You can use the PositionOnRow control itself to distinguish between the selected and the unselected rows. Once a row is selected, the PositionOnRow control on that row will be in a disabled state. So you can use different images for the disabled and enabled state of the control to differentiate between selected and unselected rows.

Multi-Row Editable List Applets

This is an extension to the Editable List Applet capability. By default when an applet is rendered in the Edit List mode in a view, only the currently selected row is editable. To edit other rows, you need to save the current changes and then select the next row to edit.

It is also possible to render list applets in Edit List mode where all the rows can be edited. Users can update multiple rows and then save all the records with one invocation of the WriteRecord control.

To make a list applet support multi-row edits in Edit List mode, set the HTML Multi Row Edit property of the List object in Siebel Tools to TRUE. The default for this attribute is FALSE. All the other steps are the same as for the regular Edit List mode.

NOTE: You do not need to place the WriteRecord control on each row. Only one such control is required for the applet.

There are certain limitations around the usage of this feature:

- If an error occurs while committing any of the records, the Siebel Web Engine will try to commit as many of the records that it can and will report errors on all the failed records. However, the error messages may not have sufficient information on which rows failed.
- Changes in the current working set must be committed before you can navigate to another working set (in other words you need to save your changes before calling GotoNextSet, GotoPreviousSet, and so on).

Because of these limitations, this feature should be used only in cases where these limitations will not cause a significant impact on the application's usability.

Hence this feature should be used only when the following conditions are met:

- Validation errors in the editable fields of the applet can be caught with client side validation (using the Browser Script).

- Only one user will be updating the records of this applet at any given time.
- The number of records in the list applet are small enough that they can be rendered on a single page without the Next or Previous controls.

A good example of the use of this feature is to update the Quantity field in the Shopping Cart applet.

NOTE: This feature is specific to standard interactivity applications. Applications that use high interactivity commit implicitly as you navigate between rows of a list applet. You can edit any row of a list applet, and as you proceed these changes will be committed to the database.

Multi-Record Select List Applets

Multi-select list applets provide a way to select multiple items for a transaction. The check boxes in the left column are used to select the items. The Select All button allows the user to select all available records in the list. The Select action button selects all of the records that have been chosen for inclusion in the selection.

This feature is specific to standard interactivity applications. In applications that use high interactivity, multi-row selection is available in all list applets rendered using the `<swe:list>` tag, except for pick applets. In these applets, multiple rows can be selected using the Control/Shift keys, as in any standard Windows application

The Siebel Web Engine supports the selection of multiple records in list applets for invoking methods that act on these selected records. With the HTML Client the selection of rows is done using check boxes that are placed on each row.

This is different from positioning the current record using the PositionOnRow control. You can have both the PositionOnRow control and Multiple Row Selection on the same list applet.

When you initially navigate to a list applet, the record on which the bus comp is positioned is automatically selected. Users can unselect this using the check box if desired. Unlike PositionOnRow, when you select rows using the check box there is no server round trip. The selected records are marked as selected on the bus comp only when a method is invoked on the applet. You can select records across multiple pages (that is, you can navigate using the Next and Previous controls and select records from different working sets).

By default, multirecord selection is not enabled for list applets. To enable this feature on list applets where this needs to be supported, set the new attribute of the List object in Siebel Tools called HTML Multi Row Select to TRUE.

To render the check boxes to select multiple rows in list applet templates, the tag `<swe:select-row>` is used. The syntax of this tag is:

```
<swe:select-row property="FormattedHtml" />
```

or

```
<swe:select-row>
   <swe:this property="FormattedHtml" />
</swe:select-row>
```

When the property attribute is set to FormattedHtml in either the <swe:select-row> or <swe:this> tag, the check box will be rendered if the applet is enabled for multirecord selection in Tools. When <swe:select-row> tag is used without the property attribute, it acts as a conditional tag to show its body if the applet is enabled for multirecord selection.

By using this tag, you can create a generic list applet template that can be used with list applets that support multi-record selection and those that do not. In the list header, use the <swe:select-row> tag conditionally to put in a <td> for the header for the row selection check box column, and in the list body use the <swe:select-row> tag along with the <swe:this> tag conditionally to put in a <td> that contains the check box.

NOTE: You must place your list applet controls/list columns within a <swe:form> tag when you enable the multi-select feature, as any invoke method on the applet requires the form which contains the row selection check boxes to be submitted.

Controls that do not support invoking methods when multiple records are selected are not disabled when the user selects multiple records since there is no server call when selecting multiple records. Instead, when the control is activated a message will be shown to the user that the action cannot be performed when multiple records are selected.

Displaying Totals of List Column Values

This feature supports the following:

- Simple summation of values in a list column
- Totals based on expressions defined at the business component field level

For example, the Revenue business component has the fields Quantity, Price, and Calculated Revenue. The field Calculated Revenue has an expression defined in its Calculated Value attribute as [Quantity]*[Price]. In a list applet based on this business component, you can show the total quantity and the total revenue. The total value for quantity is the sum of all quantity field values. The total value for revenue is the product of the totals of the quantity and price columns.

To configure a list applet to display totals

1. Set the Totals Displayed and Totals Required properties of the List object to TRUE.
2. Set the Total Required property of the specific list columns that need to be totaled to TRUE.
3. Set the Web Template used by the Base or Edit List applet Web template to Applet List Totals (Base/EditList).
4. Use the value Total for the Property attribute of the <swe:control> tag in the template file:

```
<swe:control id="XXX" property="Total"/>

or

<swe:control id="XXX">

   <swe:this property="Total"/>
```

```
</swe:control>
```

When the Property attribute is set to Total, either in the <swe:control> tag or the <swe:this> tag, the total for the list column values is rendered if the list column is enabled for totals. If the list column is not enabled for totals, no output is generated. This property is valid only when the <swe:control> tag is mapped to a list column.

To enable expression-based totals

1. Check that the business component field to which the list column is mapped has an expression defined.
2. Set the Total Required attribute.
3. Add a user property named TotalAsExpr for the list column.

   NOTE: Adding the user property is enough to evaluate the totals as an expression; the fields properties are ignored.
4. Use the value Total for the Property attribute of the <swe:control> tag in the template file as described in Step 4 under "To configure a list applet to display totals" above.

You can also show totals in a separate applet. An example of showing the totals in a separate applet can be seen in the Quote Details View. A form applet appears below a list, which contains summations of columns within the list.

To show totals in a separate applet

1. Create a form applet and place it below the list applet in the view.
2. Create a field in the business component that sums a multivalued field using the calculated expression syntax Sum([Multi Value Field]).
3. Create in the business component a Multi Value Link object and a Multi Value Field object based on the link.

   The Multi Value Link object references the business component that supports the list of values to be summed.

   CAUTION: Never put a Sum([Multi Value Field]) expression in a list column. This requires that a separate query be executed for each record in the list, which is a significant performance issue.

Multi-Value Group and Pick Applet

If a control or list column has an MVG applet configured in Siebel Tools, SWE will have the following behavior:

- In Base mode, the field shows the primary value in the MVG. There is no link to pop up the MVG applet in this mode.

- In Edit, New or Edit List mode (provided the control or list column is editable), the field will show the primary value of the MVG as read-only text followed by a link to pop up the MVG applet. The style of the link to the MVG is configured using the EditFieldCaption and EditFieldType properties in the cfg file.
- When the link is activated, the MVG applet is rendered on a separate pop-up window.
- If the MVG applet has an Edit List type template defined in the repository, that template is used to render the applet. If not, the Base template is used. An error is generated if both Base and Edit List templates are not defined.
- The MVG applet behaves like any other list applet in the pop-up window. You can invoke methods like EditRecord, AddRecord, and CreateRecord. When these methods are invoked, the appropriate template is displayed in the current pop-up window. After the record is saved or selected, the MVG applet is again rendered in this window in the Base/Edit List mode.

About Tree Applets Templates

The explorer-style (or tree) applet presents hierarchically structured information in an expandable tree control. The tree control is displayed in a frame on the left side of the applet content area. Detailed information for a selected tree node is displayed in the details applet in a frame to the right. The separate vertical frames allow the contents of the tree applet to be scrolled independently from the details applet. This is important because trees' structures can typically grow very large in length and width.

A tree applet in an explorer view is similar in operation to the Object Explorer and Object List Editor in Siebel Tools. The user may expand and collapse folders in the tree applet, and view the records in that folder in the list applet. The hierarchy displayed in the tree applet represents master-detail relationships between records of different business components. A tree applet in an explorer view uses the set of master-detail relationships implemented in the business object assigned to the view. The Opportunities Explorer View is illustrated in Figure 128.

Figure 128. Opportunities Explorer View

For example, when the user expands an opportunity by double-clicking, a set of folders appears hierarchically beneath it including Opportunities, Contacts, Partners, Quotes, Activities, Notes, and so on. When the user expands one of these child folders, a list of records appears of the corresponding business component. If the user expands the opportunity and then expands the Activities folder beneath it, the list of records displayed is the set of activity records for that opportunity. In the master-detail relationship between opportunities and activities, these activity records are detail records of the master opportunity record that was expanded. The user can also add or associate detail records of various kinds to particular master records.

This section describes the configuration of the templates for the explorer applet.

Here is a sample view template for a view containing an explorer applet:

```
<!--View with tree applet on the left and list applet on the right-->

<table border=”0” cellspacing=”0” cellpadding=”1” width=”100%”>

   <tr>

      <!-- Begin Tree Applet -->

      <td>

         <swe:applet id=”1” hintText=”Tree Applet”/>

      </td>

      <!-- Begin List Applet -->

      <td>

         <swe:applet-tree-list/>

      </td>

   </tr>

</table>
```

The <swe:applet-tree-list> tag that appears in this template provides a placeholder for a list applet that is displayed as a result of selecting or expanding a tree item (node). The applet that is rendered depends on the node that is currently selected.

Here is a sample applet template for an explorer applet. It displays the tree in a single-column table:

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>

   <TBODY>

      <swe:for-each-node>

      <TR VALIGN=top>

      <TD NOWRAP>

      <swe:for-each-indent>

         <swe:indent-img/>
```

```
        </swe:for-each-indent>
        <swe:node type="DisplayName">
           <swe:this property="FormattedHtml"/>
        </swe:node>
        <swe:node type="FieldValue">
           <swe:this property="FormattedHtml"/>
        </swe:node>
        </TD>
        </TR>
        </swe:for-each-node>
    </TBODY>
</TABLE>
```

A tree control can have repository tree nodes and field values as elements in the tree. The term *tree item* is used for a tree element regardless of whether it is a "root," "branch," or a "leaf" in the tree. A repository tree node is called a *tree node*. The <swe:node> tag specifies the placeholder for a tree item. For a tree node, the display name is shown, whereas for tree items, the field values are shown. In the example above, the <swe:node> tag with type DisplayName is ignored for tree items, and the <swe:node> with type FieldValue is ignored for tree nodes.

In order to display a tree, the logic iterates over each item of the tree in a top-down, depth-first fashion, and displays one item at a time. This is specified in the template using the <swe:for-each-node> tag.

Each tree item is indented to place the text in the correct indent level relative to the root using the <swe:for-each-indent> tag, and to display the expand/collapse mark, the text of the item, and the hyperlinks. The indentation is accomplished using a series of GIF images, or just white spaces (when in text-only mode). The expand/collapse mark and the item are displayed using images (or just text, in text-only mode), specified in the template using the <swe:indent-img> tag. The list applet associated with the currently selected tree node is displayed as part of the view.

Details about the various tags used in Tree Applet Templates are described below:

- <swe:for-each-node>

  Purpose:

  Iterates over each visible item in the tree control in a top-down, "depth-first" fashion. This tag is used to display tree nodes and field values. The attributes are optional. If Count is not specified, the tag iterates over all nodes in the tree.

  Attributes:

  - **Count.** Specifies the number of times the tag should iterate its contents. This attribute is provided for situations where specific tree formatting is required.

- **StartValue.** The value at which the iteration starts. The tag starts the iteration by assigning this value to an internal iterator, and increments it by one for each iteration.

- <swe:for-each-indent>

  Purpose:

  Iterates over each level of a tree item. Used for creating indentation when displaying tree items.

  Attributes: None.

- <swe:indent-img>

  Purpose:

  Provides a placeholder for a GIF image corresponding to a tree item's current indentation level. At each level, SWE determines which GIF file to use in the <img> tag to output. The GIF images can be either a blank space or a vertical bar.

  Attributes: None.

- <swe:node>

  Purpose:

  Provides a placeholder for an item in the tree. A tree item can be a repository tree node or a field value. The display name is printed if the tree item is a tree node. Otherwise, the field value is generated. The expand/collapse mark, the item's icon, and the links are also parts of a tree item. Depending on the configuration file settings, the expand/collapse mark is shown as either a GIF image or text. The expand/collapse mark is only shown for tree items with child items. There are two links associated with each item. There is a link for the +/— mark to expand or collapse the item and a link for the item image for selecting the item (or for going to next or previous workset). The item selection allows the user to access the list applet associated with the tree node. This tag should use <swe:this> as a child tag.

  Attributes:

  - type. Set to "DisplayName" or "FieldValue". Outputs the repository tree node's Display Name if "DisplayName." Otherwise, outputs field values.

Configuration File Parameters

A tree control consists of reusable graphic elements and text obtained from a business component record, as shown in Figure 128.

The text is obtained from business components, as defined in the Tree and Tree Node object types in the repository. The graphic elements (expansion and contraction boxes, elbows, folder symbols and so on) are defined in the configuration file in the [SWE] section. Configuration file parameters are specified to customize the appearance of the folder and document symbols, expand and collapse marks, elbows, spacers, and so on. The syntax of a configuration file parameter line for defining a graphic is as follows:

```
parameter_name = <img param1 param2 etc.>
```

For example:

```
TreeNodeCollapseCaption = "<img src='images/tree_collapse.gif' alt='-' border=0
align=left vspace=0 hspace=0>"
```

A text replacement for an image, for use by text-only browsers, is specified using the `alt=` parameter in the `<img>` tag.

Four parameters are also available for configuring the display of text obtained from field values. These are the TreeNodeFontStyle, TreeNodeFontSize, TreeNodeSelectBgColor, and TreeNodeSelectFgColor parameters. The syntax for these is:

```
parameter_name = value
```

The term *caption* as used in the parameter names actually means icon or graphic. The caption (image) precedes the text that is generated from field values, or precedes another caption. The supported caption graphic and text style parameters are listed below by category.

Elbows and Trees

- TreeNodeCollapseElbowCaption
- TreeNodeCollapseTeeCaption
- TreeNodeElbowCaption
- TreeNodeExpandElbowCaption
- TreeNodeExpandTeeCaption
- TreeNodeTeeCaption

Root, Leaf, and Open/Closed Folder Icons

- TreeNodeCloseFolderCaption.
- TreeNodeLeafCaption.
- TreeNodeOpenFolderCaption—Open folder with the dangling line.
- TreeNodeOpenFolder2Caption—Open folder without the dangling line.
- TreeNodeRootCaption.
- TreeNodeArrowDownCaption—This icon indicates that there are additional records not shown below, and when clicked, displays the next group.
- TreeNodeArrowUpCaption—This indicates that there are additional records not shown above.

Indentation Graphics

- TreeNodeBarCaption.
- TreeNodeSpaceCaption.

Text Style Parameters

- TreeNodeFontStyle. Defaults to MS Sans Serif,Arial,Helvetica.

- TreeNodeFontSize. Defaults to 1.
- TreeNodeSelectBgColor. Defaults to #000080.
- TreeNodeSelectFgColor. Defaults to #ffffff.m

About Chart Applet Templates

Chart Applets display business component data as different types of charts and graphs. Templates for charts contain a handful of swe:control tags to map the Chart Control (id=599) in the standard configuration) and the various controls for switching between different chart types, and so on. A typical chart template is shown below:

```
<table width="98%" cellspacing="0" cellpadding="0" border="0" align="center">
<swe:form>
   ...
<table width="100%" valign="top" align="center">
   <swe:togglebar type="Select">
   <tr>
      <td>
         <swe:control id="2" property="DisplayName" />
      </td>
      <td>
         <swe:this property="FormattedHtml"/>
      </td>
   </tr>
   </swe:togglebar>
</table>
   ...
<table class="AppletBack" width="100%" border="0">
   <tr>
      <td align="center">
         <swe:control id="599" property="FormattedHtml" hintText="Chart"/>
      </td>
   </tr>
```

```
</table>

   ...

</swe:form>

</table>
```

About Catalog-Style List Applets and Rich List Templates

This feature supports a catalog-like layout for views with master-detail applets. Records from the master applet and the detail applet can be shown interwoven with each other. This allows the creation of the layout like the one shown below in Figure 129:

Figure 129. Master-Detail Applet

In this case the bullet items under Portable Music are from records in the master applet. The values below it are records from the detail applet for that record in the master applet.

To create this layout, the master and detail applets are configured to be list applets. The master applet will be called a root level applet. It is possible to show more than one set of master-detail relationships within a view (that is, there could be more than one root level applet). To define the relationship between the applets, the new Position attribute of the View Web Template Item object type is used. The position attribute works similarly to the Position attribute of the Tree Node object type. The root level applets will have position values like 1, 2, and so on. For the applet with position 1, its immediate child applets will be assigned position values 1.1, 1.2, and so on. It is possible to define third level applets with position 1.1.1, 1.1.2, and so on (that is, these are the child applets of the applet with position 1.1).

In the View Web Template Item object definition, only the root level applets are mapped to `<swe:applet>` tags in the view template. The other applets in the view defined in the View Web Template Item object are not assigned an Id value. The layout of these non-root applets are not specified in the view template, but are specified in the applet template of the root level applets. The following tags are used to specify this layout. Only applets in the base mode in this layout are supported.

- swe:for-each-child

  Usage:

  <swe:for-each-child> ... </swe:for-each-child>

  Purpose: This tag iterates over each of the child applets defined for this applet (based on the Item Identified in the View Web Template Item object of the view to which the applet belongs). This tag can be used only in the base template of an applet. If the applet does not have any child applets, this tag is skipped.

- swe:child-applet

  Usage:

  <swe:child-applet/>

  Purpose:

  This tag is used to place the child applet within the parent applet. The base template of the child applet is used to render the child applet at the point where this tag is placed.

Example

This section presents a master-detail applet relationship with the master applet being Category Items List Applet and the detail being Sub Category Items List Applet. The View Web Template Item of the view that contains this applet has the following values (Table 67):

Table 67. View Web Template Item Properties

Item Identifier	Applet	Applet Mode	Position
101	Category Items List Applet	Base	1
	Sub Category Items List Applet	Base	1.1

The base template for the Category Items List Applet will have the following table definition:

```
<table>
<swe:for-each-row>
<tr>
   <td>
      <swe:control id = "5001" /> <!-- field value like "Small Business" -->
   </td>
   <td>
      <swe:for-each-child>
```

```
            <swe:child-applet> <!-- Show the child applet -->
        </swe:for-each-child>
    </td>
</tr>
</swe:for-each-row>
</table>
```

The base template for the Sub Category Items List Applet will have the following:

```
<table><tr>
<swe:for-each-row>
    <td>
        <swe:control id="5001"/> <!-- field value like "Desktop" -->
    </td>
</swe:for-each-row>
</tr></table>
```

NOTE: Set the HTML Number of Rows property of the Sub Category Items List Applet to the number of values you want to show under each category value. To allow drilldown from the category and sub-category values, configure the appropriate drilldown objects.

About Siebel Tags

Siebel tags are special tags you insert into template files. They specify how objects defined in the repository should be laid out and formatted in the final HTML page in the user's Web browser.

The process of configuring a Web application separates the layout and formatting from the application definition and binding to data. You use Siebel tags to determine the layout and formatting of controls in your application.

How Siebel Objects are Mapped to IDs in Web Templates

The .SWT template files do not include references to specific controls in the repository. Instead, they specify a layout and style, with placeholder tags. The following is an example of a Siebel tag that places a Web Page Item in a Web Page. Other Siebel tags place other items in a Web page, such as view bars, applets, or controls.

```
<swe:Control id="1" property="FormattedHtml"/>
```

To process this tag and generate the final HTML the Siebel Web Engine does the following:

1. Examines the compiled .srf file for the properties of the Web Page Item in the current Web Page that has an Item Identifier equal to 1. This is the mapping between the template file object and repository object.
2. Renders the Formatted HTML representation of this repository object in place of the abstract placeholder in the template file.

Figure 130 shows how the mappings between controls and IDs work for displaying an image as a link to add a new contact. The example illustrates the general point, but note that in your actual implementation, the HREF probably will not look exactly like this. If you create the right controls and template mappings, Siebel Web Engine will construct a URL in the HREF that will execute the method NewRecord in the correct context.

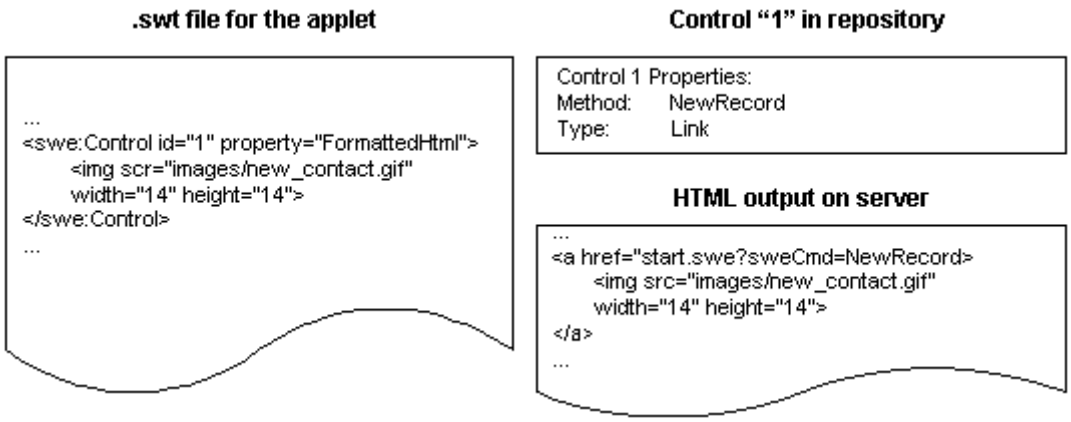


Figure 130. Mappings Between Controls and IDs

About Singleton and Multi-Part Tags

Singleton and multi-part tags are part of the basic vocabulary of SGML, so they are only discussed here to introduce the concepts and terminology. Siebel eBusiness Applications use singleton and multi-part tags in the standard way.

A *singleton* element is a tag that includes the end-tag slash in the same tag as the tag name. There are no child elements in a singleton tag. The following is an example of a singleton tag:

```
<swe:pageitem name="value"/>
```

The following is an example of a multi-part tag because it does not have the end-tag slash:

```
<swe:control id="1" property="formattedHTML">
    ...HTML here...
</swe:control>
```

About the "This" Tag

Sometimes you will want to use a multi-part tag, but make reference to the SWE-generated control at some point other than the beginning and end of the tag. To do this, you can use a "this" tag:

```
<swe:control id="1">
    ...HTML here...
    <swe:this property="formattedHTML"/>
</swe:control>
```

The <swe:this> tag is an alias for the nearest enclosing Siebel context. Often, this context is established by an enclosing <swe:xxx> element. For example, <swe:this> commonly appears inside a multi-part <swe:control> element. In that case, <swe:this> is an alias for the control. It is used to display properties of the control. In some cases, the context is less direct. For example, if an <swe:this> element appears in an applet template file, outside of any <swe:control> tag, it is an alias for the applet, and can be used to display properties of the applet.

About Iterators

Iterator tags specify the number of times the tag should iterate its contents. For example, the swe:for-each tag allows you to reduce the size of the template files where the same HTML and Siebel tags are used with controls or page items with different values for the id parameter:

```
<swe:for-each count="x" iteratorName="yyyy" startValue="z"/>
```

Other iterator tags include swe:for-each-row, swe:for-each-child, swe:for-each-node, swe:for-each-indent, swe:for-each-value.

The attributes of the swe:for-each tag are as follows:

- **count.** Specifies the number of times the tag should iterate its contents.
- **startValue.** The value that should be assigned to the iterator at the start of the iteration. The tag will start the iteration by assigning this value to the iterator, and will increment it by one for each iteration.
- **iteratorName.** The name of the iterator. This name can be used to get the value of the iterator during the iteration using the syntax swe:iteratorName.

In the section enclosed by the swe:for-each tag, references to the current value of the iterator is through the name specified in the iteratorName attribute. For example, if you set the value of the iteratorName to "CurrentID," then you can get the value of the iterator using the syntax swe:CurrentID. You can also reference a value that is an increment over the current value as swe:CurrentID+x. The fragment below illustrates this usage:

```
<swe:for-each startValue="2301" count="50" iteratorName="currentId">
  <swe:control id="swe:currentId">
  .
```

```
    .
    </swe:control>
    <swe:control id="swe:currentId+100" />
  </swe:for-each>
```

About Nesting and Siebel Tags

You cannot nest a Siebel tag inside HTML tags. For example, the following is not valid and will generate an error:

```
<img src="<swe:control id="1"/>">
```

In addition, you cannot nest some Siebel tags. For example, the following is not valid and will generate an error:

```
<swe:control id="1">
   <swe:control id="2" property="formattedHTML"/>
   <swe:this property="formattedHTML"/>
   </swe:control>
</swe:control>
```

About SWE Conditional Tags

The SWE framework supports the <swe:if> conditional tag, which provides a simple conditional branching capability. The <swe:if-var> tag is a variation on <swe:if> that permits you to evaluate a namespace within an applet template. For further information see *Siebel Developer's Reference*.

About Using Toolbars and Menus in Templates

Toolbars allow the user to initiate various actions. Application menus (File, Edit, View and so on) appear in their own frame near the top of the application in the browser window, and the application toolbar appears just beneath it, as shown in Figure 131.

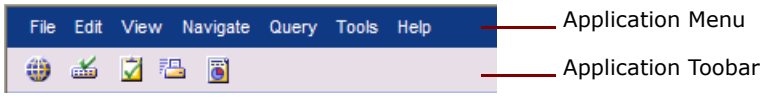


Figure 131. Application Menu and Toolbar

The applet-level menus are invoked from the applet menu button, in the control banner at the top of an applet. This is illustrated in Figure 132 where the menu button is in the left corner of the control banner.

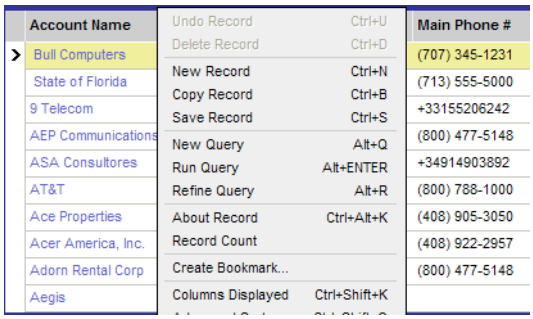


Figure 132. Applet Menu Button

Clicking on a toolbar icon or menu item is normally translated into a call to an invoke method, which may reside in a service on the browser or server, or in classes in the browser application or server infrastructure (applet or business component classes). The toolbar icon or menu item is configured to target a method name, a method handler, and optionally a service.

Application-level items (which include both toolbar icons and application-level menus) are implemented through the use of Command object definitions in Tools, which are then mapped to Toolbar Item or Menu Item object definitions. Applet-level menus do not use Command object definitions, but the set of properties used for targeting the method are essentially the same as those found in the Command object type.

How Toolbars are Displayed in Templates

Toolbars are displayed as HTML toolbars in standard interactivity applications and as extensible JavaScript toolbars in high interactivity applications. For applications using CTI, an additional toobar can be displayed as a Java applet.

HTML toolbars reside in the topmost frame in the application template, which is set aside for this purpose. JavaScript toolbar objects reside in the JSSApplication hidden frame, which usually does not reload during the application lifecycle. Therefore, they are not redrawn when there is a page refresh. The UI part of the JavaScript toolbar resides in a visible HTML frame (recommended to be a persistent frame that reloads infrequently) and redraws when the HTML frame reloads.

In SWE templates, the `<swe:toolbar>` tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the `<swe:toolbaritem>` tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository.

HTML and JavaScript Toolbars

For an HTML or JavaScript toolbar, add the following to the SWT file:

```
<swe:toolbar name=xxx>   // where xxx is the name of toolbar in the repository.
   // any HTML stuff here...
      <swe:toolbaritem>
   // any HTML stuff here...
</swe:toolbar>
```

NOTE: For combobox items, the command has to be targeted to a service.

Java Toolbars

For a Java toolbar, add the following to the SWT file:

```
<swe:toolbar name="xxx" javaapplet="true" />
```

The Java applet invokes the ShellUIInit method on the command target service when it tries to initialize. It invokes ShellUIExit when it exits. There is a set of communication protocols defined for the communication between the Java Applet and the service.

The toolbar is implemented as a Java Applet (including all the toolbar controls and the threads interacting with the server).

The full syntax specifications for the <swe:toolbar> and <swe:toolbaritem> tags are below:

- <swe:toolbar>

  Purpose:

  In SWE templates, the <swe:toolbar> tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the <swe:toolbaritem> tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository. Siebel eBusiness Applications currently support two types of toolbars: HTML toolbars and Java applet toolbars, as specified in the javaapplet attribute.

  Usage:

  ```
  <swe:toolbar name="XXX" javaapplet="true/false" width="XXX" height="XXX" />
  ```

  Attributes:

  - **name.** The name of the toolbar as defined in Tools.
  - **javaapplet.** This should be set to true for java toolbar, and false for HTML toolbar.
  - **width**. Width of the toolbar in pixels.
  - **height.** Height of the toolbar in pixels.

- <swe:toolbaritem>

  Usage: <swe:toolbaritem>

  Attributes: None.

How Menus are Displayed in Templates

A menu, in SWE templates, is a button or link that activates a drop-down list of menu selection items when clicked by the user. There are application-level and applet-level menu buttons/links. The <swe:menu> tag will render all menus that are appropriate at the template level where it occurs—application (container page) or applet. Menus are shown only when running in the High Interactive mode. An applet menu is rendered as an icon button, generally placed to the left of the other buttons such as Edit and Delete. At run time, the set of menu options for a given applet are generated from the .srf file. Note that for applet level menus, the tag must be specified in an applet template.

A sample of the template for rendering an applet's buttons, including the menu button, appears below (from CCFormButtons.swt):

```
<!-- Buttons (Edit, Delete, Optional, Optional, Optional) --->

<!-- Menu,179 -->

   <td valign="middle" nowrap>

      <swe:menu/>

   </td>

   <td valign="middle"><img src="images/bttns/div.gif" height="13" border="0"
   align="middle" alt=""></td>

<!-- EditRecord -->

<swe:control id="132">

   <td valign="middle" nowrap>

      <swe:this property="FormattedHtml" hintText="Edit" hintMapType="Control"/>

   </td>

   <td> </td>

</swe:control>

(and so on...)
```

For the application level menu, the <swe:menu> tag can be specified in any template other than an applet template. The set of menus is rendered from the one <swe:menu> tag, based on the Menu and Menu Item object definitions in the repository. The Menu object definition that is used is the one pointed to by the Menu property in the Application object definition. This Menu object definition specifies a set of top-level menu names and menu options within each top-level menu. The Profile and Logout menu options are automatically provided in all applications, and do not have to be defined within the application's Menu object definition and its children.

The following code sample (from CCFrameBanner.swt) illustrates the inclusion of the <swe:menu> tag at the start of the definition for a banner:

```
<!--Start Banner-->

<swe:menu/>
```

```
<table class="banner" cellpadding='0' cellspacing='0' border='0'>
<tr>
    <td width="50%">
        <img src="images/SiebelLogo.gif" width="282" height="57" border="0" alt="" />
    </td>
    <td width="50%">
        <img src="images/spacer.gif" width="100%" height="58" border="0" alt="" />
    </td>
</tr>
```

The syntax for the <swe:menu> tag appears as follows:

- <swe:menu>

  Purpose:

  Renders menu buttons or links for all menus defined for the relevant entity, either an application or an applet. For an application, one button or link is rendered for each top-level menu defined for the application (in its associated Menu object definition and children). For an applet, one button is rendered, the applet menu button.

  Usage:

  ```
  <swe:menu type="XXX" bitmap="XXX" width="XXX" height="XXX" bgcolor="XXX"
  fgcolor="XXX" />
  ```

  Attributes:

  - **type.** The type can be set to Default (which is the default value if this attribute is not specified) or Button. If set to Default the menu is rendered showing the top-level menu entries (like File Edit Help). If set to Button, a button is created that, when activated, shows a drop-down menu with the top-level menu entries.
  - **bitmap.** This attribute is used only when the Type attribute is set to Button. This attribute is used to specify the name of a bitmap object to be used as the label for the button. This bitmap is defined in Tools under the bitmap category HTML Control Icons.
  - **width.** This attribute is used to specify the width of the menu in pixels.
  - **height.** This attribute is used to specify the height of the menu in pixels.
  - **bgcolor.** This attribute is used to specify the background color of the menu. The color should be specified using the hexadecimal triplet format used in HTML like #FFFFFF.
  - **fgcolor**. This attribute is used to specify the foreground color of the menu. The color should be specified using the hexadecimal triplet format.

About Using the Thread Bar in Templates

The thread bar is used to track user navigation among the views. A thread bar in HTML text format has been implemented. An example of the thread bar is as follows:

```
Home >  Consumer:PCs > PCs:Laptops > Laptops:Pentium III
```

Home, Consumer:PCs, and so on are the thread buttons. The thread buttons are displayed in title: value format, and either title or value can be omitted when appropriate. The thread button may contain a hyperlink, which leads the user to a previous page. The thread buttons are separated by separators— ">" in the above example.

A thread button may have a hyperlink that leads the user to a previous page. The hyperlink requires a new SWE Command: GotoBookmarkView. The hyperlink for each thread button should contain at least the following parameters:

```
SWECmd=GotoBookmarkView&SWEBMCount=2SWECount =3
```

The SWEBMCount = 2 indicates that bookmark #2 will be used to create the view. SWECount=3 is the bookmark ID for the current view. With the definition of the swe tags and thread link format, a thread button for account A.K. Parker will be translated into HTML format as:

```
<a href = " www.siebel.com/start.swe?SWECmd=GotoBookmarkView&SWEBMCount=2&
SWECount=3> Account: AK Parker </a>
```

A new bookmark is created when the user clicks the thread button and brings back a bookmarked view. The bookmark ID for the new view is the current swe count (the count passed to the server in the request) increased by 1.

Bookmark deletion policy is not modified with the above bookmark ID assignment policy. By default, the system keeps the most recently created 20 bookmarks and deletes previous ones. If the swe count in the user request is less than the swe count on the server side, all the bookmarks with a swe count larger than what is in the user request is deleted.

The behavior of the HTML threadbar is summarized below:

- When a new screen is requested, a new thread is created to replace the current thread.
- When a view button is clicked, the last thread step is replaced by that of the new view requested.
- When the user follows a drilldown link, a new step is appended on the thread bar for the view requested.
- When a thread button is clicked, all the thread buttons to the right of it are all deleted.
- Some views may not have a thread applet or thread field defined. Showing these views do not cause the thread button to be updated.

When a thread button is clicked, the thread proceeds to the step view indicated by SWEBMCount.

The following three swe tags are defined to create an HTML thread bar. The usage of these swe tags is very similar to that of the screen bar and view bar tags.

- <swe:threadbar> Indicates the start and finish of the thread bar section.
- <swe:threadlink> Indicates the definition of a thread button on the thread bar. This tag has two properties defined:

- FormattedHtml property. Indicates that HTML hyperlink should be included.
- Title property. Indicates that the title/value pair of the thread button should be displayed.

■ <swe:stepseparator> Specifies the symbol used to separate thread buttons.

The <swe:threadlink> and <swe:stepseparator> tags should only be used within the <swe:threadbar> tag.

To use a thread bar, insert thread bar definitions into an appropriate SWT file by using the tags defined above. An example is given below:

```
<!-- Begin Threadbar section -->

<table class="Theadbar" width=100% border="0" cellspacing="0" cellpadding="0">

<tr valign="left">

   <td nowrap bgcolor="#6666CC" width=110>

      <img src="images/spacer.gif" width="110" height="25" border="0">

   </td>

   <td width=99%>

      <swe:threadbar>

         <img src="images/spacer.gif" width="10" height="25" border="0"
         align="absmiddle">

         <swe:threadlink property="FormattedHtml">

         <font color="#000000"> <span > <nobr><swe:this property="Title"/></
         nobr> </span> </font>

         </swe:threadlink>

         <swe:stepseparator>&gt;</swe:stepseparator>

      </swe:threadbar>

      <img src="images/spacer.gif" width="10" height="25" border="0"
      align="absmiddle">

   </td>

</tr>

</table>

<!-- End Threadbar section -->
```

This creates a thread bar as shown below:

```
Home > Consumer:PCs > PCs:Laptops
```

For applications without frames, put the definition in a container page such as CCPageContainer.swt; for applications with frames, insert it in the "Viewbar" frame swt file or View frame swt file.

24 Specialized Behavior Supported by Web Templates

Topics in This Chapter

About Search and Find in SWE Templates

The Siebel Web client provide users with multiple methods of locating records by merging the functions of search, find, and query in a unified search model. You can configure templates to display application-level search and query features and applet-level query features. Specialized Siebel tags are used to render the Search and Find applet, and the Results applet.

About Search and Find Applet Tags

Search and find tags are used to display the Basic and Advanced Search or Find applets. The tags are <swe:srchCategoryList>, <swe:srchCategory>, <swe:srchCategoryText>, and <swe:srchCategoryControl>.

Example*:*

```
<swe:srchCateogryList>
    <swe:srchCategory>
        <td><swe:srchCategoryText/></td>
        <td><swe:srchCategoryControl/></td>
    </swe:srchCategory>
</swe:srchCategoryList>
```

The syntax for each tag is described as follows:

"Search Tag: <swe:srchCategoryText>" on page 531

"Search Tag: <swe:srchCategoryControl>" on page 531

About Search Result Applet Tags

Search results tags are used to display the Search/Find results list applet. Search results tags appear in the CCListBodySearchResults.swt and dCCListBodySearchResults.swt templates, and include the following tags: <swe:srchResultFieldList>, <swe:srchResultField>, and <swe:this>.

Example:

```
<swe:srchResultFieldList>

   <swe:srchResultField><td align="swe:this.TextAlignment" class="Row"><swe:this
   property="FormattedHtml"/> </td>

   </swe:srchResultField>

</swe:srchResultFieldList>
```

The syntax for these tags is described in the following:

"Search Result Tag: <swe:srchResultFieldList>" on page 531

"Search Result Tag: <swe:srchResultField>" on page 531

"Search Result Tag: <swe:this>" on page 531

Search Tag: <swe:srchCategoryList>

Purpose: An iterator tag that encloses all the search categories that need to be displayed.

Usage: <swe:srchCategoryList> … </swe:srchCategoryList>

This tag establishes context and encloses the following tags:

- "Search Tag: <swe:srchCategory>" on page 530
- "Search Tag: <swe:srchCategoryText>" on page 531
- "Search Tag: <swe:srchCategoryControl>" on page 531

Search Tag: <swe:srchCategory>

Purpose: Represents a search category object.

Usage: <swe:srchCategory> … </swe:srchCategory>.

This tag encloses the following tags:

- "Search Tag: <swe:srchCategoryText>" on page 531
- "Search Tag: <swe:srchCategoryControl>" on page 531

Search Tag: <swe:srchCategoryText>

Purpose: Displays the "display name" of the search category.

Usage: `<swe:srchCategoryText/>`

This tag can be called only within the context of a "srchCategory."

Search Tag: <swe:srchCategoryControl>

Purpose: Displays the control of the search category. In the case of the Advanced Search, it is a check box.

Usage: `<swe:srchCategoryControl/>`

This tag can be called only within the context of a "srchCategory."

Search Result Tag: <swe:srchResultFieldList>

Purpose: An iterator tag that encloses all the search result fields that are defined in Tools under a Search Engine Object. The result fields are created dynamically in the buscomp and then displayed on the applet.

Usage: `<swe:srchResultFieldList> … </swe:srchResultFieldList>`

This tag establishes a context and encloses the following tags:

- "Search Result Tag: <swe:srchResultField>" on page 531
- "Search Result Tag: <swe:this>" on page 531

Search Result Tag: <swe:srchResultField>

Purpose: Represents a result field object.

Usage: `<swe:srchResultField> … </swe:srchResultField>`

This tag can be called only within the context of the srchResultFieldList and encloses the following tag:

- "Search Result Tag: <swe:this>" on page 531

Search Result Tag: <swe:this>

Purpose: Depending on the property= setting, retrieves either the text alignment setting or the value for the current result field.

Usage: `<swe:this/>`

Attributes: behavior varies depending on the enclosing context.

- property="TextAlignment." Retrieves the text alignment property for the result field from the Tools object "Search Definition - Custom result Field."
- property="FormattedHtml." Retrieves the value for the current result field from the results obtained by executing the search on the search adapter.

Favorites (Predefined Queries)

The Siebel tag <swe:PDQbar> is used to implement PDQ functionality. It has no parameters and can be located anywhere in the application. The user selects the query to be executed. The only thing the template developer needs to explicitly provide besides the <swe:PDQBar> tag is the "Favorites" label to the left of it. Ideally, the Favorites label should be implemented as a control rather than HTML text, so that it will be translated for localized or multilingual applications.

The <swe:pdqbar> tag is not required to be in the view frame. The <swe:pdqbar> tag should be placed either in the view frame or the view bar frame in applications that use HTML frames.

Query Management Commands

Users add their named queries into the combo box by means of the query management commands available as invoke method calls through the base applet classes. These are made available to the user as menu options or toolbar buttons. The following commands are supported:

- **New.** SWEMthdNewQueryE. This command places the applet in new query mode.
- **Refine.** SWEMthdRefineQueryE. This command places the applet in query-refinement mode.
- **Save.** SWEMthdSaveQueryE. This command saves the current query as a named query using its current name.
- **Save As.** SWEMthdSaveQueryAsE. This command opens up a dialog box to save the current query as a named query using a user-specified name.
- **Delete.** SWEMthdDeleteQueryE. This command opens up a dialog box to delete one of the named queries.

NOTE: Siebel Systems does not recommend implementing an Edit button for the predefined query (PDQ) feature. To implement the Edit button you would need to have it call the Refine invoke method. However, there can be problems associated with implementing an Edit button in a multiview environment, in which there is no way to determine the active view.

About Siebel Conditional Tags

The SWE framework supports the following conditional tags:

"Conditional Tag: <swe:if>" on page 533

"Conditional Tags: <swe:switch>, <swe:case>, and <swe:default>" on page 533

"Conditional Tag: <swe:if-var>" on page 534

Conditional Tag: <swe:if>

Purpose: Provides a simple conditional branching capability.

Usage: <swe:if condition="xxx"> ... </swe:if>

Attributes:

- **Condition.** The condition to check for. If the condition evaluates to TRUE, the body of the <swe:if> tag is processed. If the condition evaluates to FALSE, the body of the tag is skipped.

  NOTE: This tag does not provide an "else" capability like the if tags in programming languages. To get that behavior use the tags <swe:switch>, <swe:case>, and <swe:default> described below.

Conditional Tags: <swe:switch>, <swe:case>, and <swe:default>

Purpose: These three tags are used together to provide a conditional branching capability similar to the switch, case, and default statements in JavaScript. The <swe:switch> is a container tag for the <swe:case> and <swe:default> tags. Anything other than <swe:case> and <swe:default> within the body of the <swe:switch> tag is ignored. The condition to check is specified as an attribute of the <swe:case> tag. The <swe:case> tags are checked starting from the first <swe:case> tag. If any of the <swe:case> tags satisfies the condition, the other <swe:case> tags and the <swe:default> tags are skipped. If none of the <swe:case> tags satisfy their condition, the body of the <swe:default> tag is processed. There should only be one <swe:default> tag within the body of a <swe:switch> tag.

Usage:

```
<swe:switch>
    <swe:case condition="xxx">
        ...
    </swe:case>
    <swe:case condition="yyy">
        ...
    </swe:case>
    <swe:default>
        ...
    </swe:default>
</swe:switch>
```

Attributes:

- **Condition.** Supported only in the <swe:case> tag. If the condition evaluates to TRUE, the body of the <swe:case> tag is processed. Any subsequent <swe:case> tags within the <swe:switch> tag is skipped without checking their associated conditions. If the condition evaluates to FALSE, the body of the tag is skipped.

Conditional Tag: <swe:if-var>

Purpose: The <swe:if-var name="[value]"> tag is used within applet templates to conditionally express its body based on a variable set in a parent view template. When an applet is associated with a view, the applet's templates act as a child of the view's template for the purposes of the swe:if-var tag. The applet placeholder in the view template must specify a variable for the swe:if-var tag in the child applet template to evaluate. The swe:if-var expression returns true or false depending on whether the variable it is evaluating is a property of the <swe:applet> tag in the corresponding view template. This construct is useful for conditionally displaying parts of an applet depending on its position within a view.

Figure 133 shows a diagram of the relationships among objects and template files.

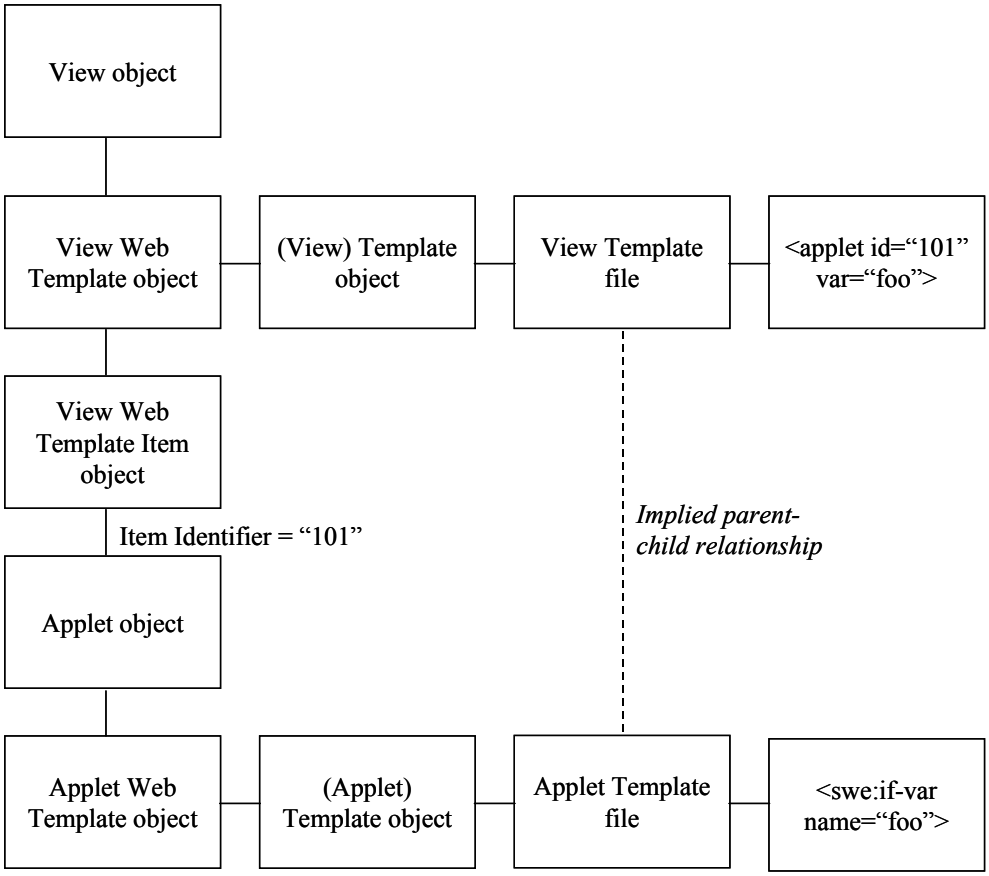


Figure 133. Object Relationships

Usage: consider an example where a view uses a template that contains the following tags:

```
<swe:applet hintMapType="Applet" id="1" property="FormattedHtml"  hintText="Applet"
var="Parent"/>

<swe:applet hintMapType="Applet" id="2"  property="FormattedHtml"
hintText="Applet" var="Child"/>
```

The view object also references an applet (through a view Web template item) whose template includes the following tags:

```
<swe:if-var name="Parent">

   <td valign="middle" nowrap>

      <swe:menu type="Button" bitmap="MenuBttn" width="38" height="15"
      bgcolor="gray" fgcolor="blue"/>

   </td>
```

```
</swe:if-var>

<swe:if-var name="Child">

   <td valign="middle" nowrap>

      <swe:menu type="Button" bitmap="MenuBttn" width="38" height="15"
      bgcolor="gray" fgcolor="red"/>

   </td>

</swe:if-var>
```

If the user drags and drops the applet into the placeholder in the view template with an id=1, the first swe:if-var condition will return TRUE and the second will return FALSE. This is because the <swe:applet> placeholder with an id=1 has its var property set to "Parent." As a result, the button menu will be displayed with a foreground color of blue. By contrast, if the user had mapped the applet to the placeholder represented by <swe:applet id="2">, the reverse would be true, and the button menu will be displayed with a foreground color of red.

About Browser Group-Specific Templates

The SWE framework supports a set of browser group-related conditions that can be checked in the Web templates using Siebel tags <swe: if> condition tag. This allows you to display different sections of a template based on which browser is used to access the application. The conditions for the <swe:if> tag are in the format <service>, <method>, <args> ...

The information about the supported user agents is defined in the Web Browser Administration views. Siebel applications have a series of predefined browsers and their associated capabilities. An example of capabilities includes FrameSupport, which indicates that a browser can support frames. Customers can modify the records that define these browsers and their capabilities as new browsers or new versions of existing browsers are introduced. Details on how to do this are provided in the *Applications Administration Guide*.

NOTE: Practically speaking the term "User Agent" is a synonym for "Browser." Its usage comes from the User Agent header property of an HTTP request which provides a unique identifier for the type of client that is making the request, such as "Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 4.0)" for Microsoft Internet Explorer 5.0.

For more information about <swe:if>, see "Conditional Tag: <swe:if>" on page 533.

Example of Checking for a User Agent

Service: Web Engine User Agent

Method: IsUserAgent

Args: UserAgent:<A User Agent name defined in the UA.INI file>

Purpose: Checks for a particular User Agent.

Example:

```
<swe:if condition="Web Engine User Agent, IsUserAgent, 'UserAgent:MSIE 5.5'">
   ...
</swe:if>
```

The condition evaluates to TRUE for the Microsoft Internet Explorer 5.5 browser and FALSE for all other browsers.

Example of Checking for User Agent Capabilities

Service: Web Engine User Agent

Method: TestCapability

Args: Capability Name:Capability Value

Purpose: Check for specific user agent capabilities. When more than one capability is provided as an argument, the condition evaluates to TRUE when the user agent has all these capabilities (AND operation).

Example:

```
<swe:if condition="Web Engine User Agent, TestCapability, 'JavaScript:TRUE',
'JavaApplets:TRUE'">
   ...
</swe:if>
```

The condition evaluates to TRUE for any user agent that supports JavaScript and Java Applets in the browser.

Example Microsoft Internet Explorer Capabilities

The following example shows the capabilities associated with Microsoft Internet Explorer:

```
[IE 5.0]
CookiesAllowed=TRUE
HighInteract=TRUE
ActiveX=TRUE
Browser=IE
Version=5
DefaultMarkup=HTML
VBScript=TRUE
JavaScript=TRUE
```

```
JavaApplets=TRUE

User-Agent=Mozilla/4.0 (compatible; MSIE 5.0

SynchExternalContent=TRUE

FramesSupport=TRUE

TablesSupport=TRUE
```

Below is an example of the extended sections for Microsoft Internet Explorer:

```
[MSIE 5.0]

User-Agent=Mozilla/4.0 (compatible; MSIE 5.0

Parent=IE 5.0

Accept=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*

[MSIE 5.5]

User-Agent=Mozilla/4.0 (compatible; MSIE 5.5

Parent=IE 5.0

Version=5.5

Accept=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*

XML=TRUE

WAP=FALSE

StyleSheets=TRUE

JavaScriptVer=1.3

DHTML=TRUE
```

How Hierarchical List Applets Are Rendered

The Siebel Web Engine supports rendering hierarchical list applets. These applets are used to show records that have a hierarchical relationship. These applets are modeled as list applets, and the hierarchy is implemented in the business component by setting the Hierarchy Parent Field attribute. These applets can be shown as a hierarchy in the Base and Edit List modes.

Hierarchical list applets do not need new templates. The icons required to display the hierarchy are added automatically to the first column in the list applet that is mapped to a business component field (Figure 134).

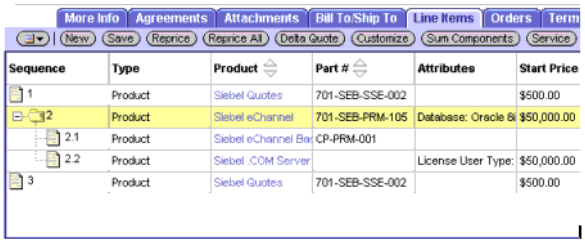


Figure 134. Hierarchical List Applet

It is also possible to show a hierarchical list applet using a Windows Explorer–like UI (very similar to the tree control). An example of this is the Categories list used to create and manage catalog categories in Siebel eSales.

The icons used for rendering the list applet should be specified in the repository using the HTML Hierarchy Bitmap object named in the HTML Hierarchy Bitmap attribute of the list object. The following bitmaps need to be specified for the HTML Hierarchy Bitmap—Expand Bitmap, Collapse Bitmap, and Space. The other bitmaps (except Arrow Down/Up Bitmap which is used only by tree controls) can be specified to get an applet that resembles the tree applet (which is not specified for hierarchical list applets in the current UI standards).

The hierarchical list applets also let you edit the hierarchy. This is done by adding controls that invoke the following methods on the applet (Table 68).

Table 68. Hierarchical List Applet Methods

Method	Action
Indent	Moves the current record to be a child of its preceding peer record (such as the record above)
Outdent	Moves the current record to be a peer of its parent record
MoveUp	Moves the current record up over its peer record
MoveDown	Moves the current record down over its peer record

When Indent and Outdent methods are invoked on a record, its relation to its child records does not change. The child records are also indented or outdented.

The MoveUp and MoveDown methods are used to position a record to invoke the Indent method. The changes made by the MoveUp and MoveDown methods are temporary; they are not saved to the database.

For a better appearance, the number of columns displayed in a hierarchical list applet should be small since the width of the column with the expand/collapse controls expand as a user navigates down the hierarchy. Similarly, the column with the expand/collapse controls should be a column with short field values.

About Browser-Specific Mappings

Different browsers can have different associated capabilities as defined in the Web Browser Administration view. Differences in the abilities of particular browsers to use features such as frames and JavaScript may necessitate different applet layouts for different classes of browsers. The browser group-specific mappings feature allows this kind of browser-specific layout customization.

A Target Browser combo box appears in the toolbar when the View > Toolbars > Configuration Context option is activated. The first item in the combo box is Target Browser Config, which activates a dialog box for selecting browsers.

The Target Browser Configuration dialog box lists all of the browsers in the Available browsers selection box at the upper left. You select the ones you are interested in working with by moving the names of these browsers into the Selected browsers for layout editing section at right. You can also view various features of the selected browser in the Capability Name and Value box in the lower half of the dialog box. The boxes in the dialog box include the following:

- Available browsers: List of available browser groups.
- Selected browsers for layout editing: Specifies which browser groups are affected by subsequent layout editing in the Web Layout Editor.
- Capability Name and Value: Specifies what capabilities or properties the currently selected virtual browser group has.

The target browser group determines how conditional template tags are expressed in the layout editor.

Inside the template are SWE:IF tags that conditionally execute blocks of code. (Markup is included.) At edit time you see the applet the way it would show up for the particular browser you have chosen. At run time, the conditional sections in the template are executed appropriately for the current browser.

For example, a user may associate a template to a view that contains the following tags:

```
<swe:if condition=”Web Engine User Agent, TestCapability, 'FrameSupport:TRUE'”>

    <swe:frameset htmlAttr=”cols='33%,66%'' border='1' frameborder='Yes'”>

        <swe:frame type=”Applet” htmlAttr=”marginheight='0' marginwidth='0'
        scrolling='Auto'”>

            <swe:applet id=”101” hintText=”Applet” var=”Parent”>

            <swe:this property=”FormattedHtml”/>

            </swe:applet>

        </swe:frame>

    </swe:frameset>

</swe:if>
```

If the user opens a view in the Web Layout Editor when his or her Target Browser is set to IE 5.0, the user will see a placeholder for the applet in the frame with an underlying identifier of 101. The user can then drag and drop a particular applet to the placeholder. This is because IE 5.0 has a FrameSupport capability with a value of TRUE. However, if the user's Target Browser is set to IE 1.5, he or she will simply not see the placeholder in the layout editor since the FrameSupport capability for IE 1.5 is set to FALSE.

Creating Custom HTML Control Types

Siebel applications support the use of several different control types (for example, Check Box, Button, Mail To, Text Area, and so on). However, you may also require additional HTML types. You can define your own HTML types by adding type definitions to the .SWF file.

Unlike cascading style sheets, which are used primarily to define general stylistic information about labels, titles, background colors, and so on, user-defined types in the CCHTMLTYPE.SWF file would normally be used to define more complex attributes that determine either the appearance or client-side functionality of a type of HTML element. Examples might include a button type that is associated with a particular GIF or a type of link that connects the user with an FTP site.

Again, these attributes could also be given to a page element by specifying the appropriate tags and attributes directly in the .SWT file. However, by defining them as types within the CCHTMLTYPE.SWF file, they can be referenced from within Siebel Tools for a specific control on a specific Applet Web Template or Web Page object. This preserves the generality of .SWT templates by avoiding the need to place HTML directly within them. It also improves the maintainability of the application by minimizing customization done to templates and storing more configuration information within the repository.

NOTE: Custom control types that invoke methods are not supported in HI Interactivity.

To create and use a new HTML Type

1. Add the name of the new type (for example, "MiniButton") to the List of Values used for the HTML Type property in Siebel Tools (REPOSITORY_HTML_CTRL_TYPE).
2. Add the formatting information for the new type to one of the application's two .SWF files. These files, which should use the extension .SWF, must reside in the same directory as the template files. One file contains the special types defined by Siebel Systems, Inc.; the other contains customer definitions, either to add additional types or to override Siebel types.

   The format for rendering the custom type is specified by using two tags:

   - `<swe:htmltype>`
   - `<swe:this>`

   The details of defining a new format are explained below, but for now, the form is as follows:

```
<swe:htmltype name="XXX" mode="AAA" state="BBB">

    ..... HTML .....

    <swe:this property="YYY" />
```

```
    .... More HTML ....

</swe:htmltype>
```

3 Specify the names of the .SWF files to be used by an application in the [SWE] section of the .CFG file used by the application's object manager.

```
[SWE]

SystemSWFName = CCHtmlType.swf

UserSWFName = MyFormat.swf // You must set this name.
```

4 In Siebel Tools, change the HTML Type property of the control, list column, or page item to the new type.

5 In the template file, use the FormattedHTML property for the <swe:control> or <swe:pageItem> element.

When SWE Uses a Custom HTML Type

If the HTML type of a control, list column, or page item is a custom type, the Siebel Web Engine will use the .SWF format when rendering any elements that are mapped to the control, and that specify the FormattedHtml property.

The following cases exist:

- <swe:control id="1" property="FormattedHtml"/>
- <swe:control id="1"> ... <swe:this property="FormattedHtml"/> ... </swe:control>
- <swe:pageitem id="1" property="FormattedHtml"/>
- <swe:pageitem id="1"> ... <swe:this property="FormattedHtml"/> ... </swe:pageitem>

The formatting will not be used with any other property, such as Display Name. However, in the .SWF file, the <swe:this> element can refer to these properties (except the FormattedHtml property itself).

More About Format

The .SWF files contain multiple format specs. Each format spec includes two parts:

- An enclosing XML element that names the type and optionally names the mode and state in which the current format is used
- The enclosed format content

The format content syntax follows these rules:

- It must be valid, regular SWE syntax.
- It may refer to all the properties of the current control, except "FormattedHtml" (in order to prevent recursion).
- It may use a new swe:this property, "Data," which is explained below.

Examples

Example 1: To create a new HTML type for a control called "LabelRed" that shows the caption of the control in red, the formatting might look like this.

```
<swe:htmltype name="LabelRed">

   <font color="red"> <swe:this property="DisplayName"/> </font>

</swe:htmltype>
```

Example 2: The Data property of <swe:this> is something like a macro that casts the current, custom type to one of the intrinsic types, and then inserts the FormattedHtml property of the intrinsic type. To use the Data property, you add an additional Type attribute to the <swe:this> tag, that names the intrinsic type. For example, to create a new type called "MiniButton," in which special formatting is added to the Web Engine intrinsic type Link, you might write a format something like this:

```
<swe:htmltype name="MiniButton">

   <img SRC="images/btn_left.gif" border=0 height=15 width=2>

   <swe:this property="Data" type="Link" />

   <img src="images/btn_right.gif" border=0 height=15 width=2>

</swe:htmltype>
```

Here the <swe:this property="Data" type="Link" /> will output the same HTML as if the template included a separate <swe:this> tag, where the property was FormattedHtml, and the HTML type of the control were the built-in type Link.

You can only specify built-in types and not custom types for the type attribute of Data elements.

Example 3: You can also define custom formats for the different applet modes (Base, Edit, New, Query), by using the Mode attribute for the <swe:htmltype> tag. If a mode is specified, then that formatting is used only if the current show mode matches the value specified for this attribute. For example, if you want to create a new HTML type called SiebelText to show a control which displays as a label and a text field while in Edit template, and as read-only text in Base mode, you specify the format as:

```
<swe:htmltype name="SiebelText">

   <swe:this property="Data" type="Text"/>

</swe:htmltype>

<swe:htmltype name="SiebelText" mode="Edit">

   <swe:this property="DisplayName"/>: <swe:this property="Data" type="Text"/>

</swe:htmltype>
```

Example 4: You can define another optional attribute to the <swe:htmltype> tag, namely State, to show different formatting depending on the state of the control or list item. Currently Siebel Web Engine supports two states:

- **Disabled.** For controls or list columns that invoke methods, where the method cannot be invoked on the record.
- **Required.** For controls or list columns that are required.

For example, to show grayed-out buttons when a method cannot be invoked, add the following format definition in addition to the default definition shown earlier.

```
<swe:htmltype name="MiniButton" state="Disabled">

   <img src="images/graybtn_left.gif" border=0 height=15 width=2>

   <swe:this property="Data" type="Link" />

   <img src="images/graybtn_right.gif" border=0 height=15 width=2>

</swe:htmltype>
```

With built-in HTML types, if a method cannot be invoked, then the control or list item is not shown (same as the current behavior). With custom HTML types, however, the formatting specified in the .SWF file is always shown. The HTML generated for the property Data (`<swe:this property="Data" type="Link" />`) when the method cannot be invoked is simply the caption of the control or list item without any `<a href>` tags.

You can hide a control or list item with a custom HTML type when a method cannot be invoked by creating an empty `<swe:htmltype>` tag for the Disabled state.

```
<swe:htmltype name="MiniButton" state="Disabled"></swe:htmltype>
```

NOTE: This only hides the `<swe:control>` or `<swe:this>` tag that invokes the FormattedHtml property.

Example 5: To show the SiebelText type with an indicator (*) for required fields you can add the following format definition in addition to the definitions for this type shown earlier.

```
<swe:htmltype name="SiebelText" mode="Edit" state="Required">

   * 

   <swe:this property="DisplayName"/>

   : 

   <swe:this property="Data" type="Text"/>

</swe:htmltype>
```

NOTE: When SWE looks up HTML Type definitions in the .SWF file, the order of precedence will be Mode and then State. It is recommended to always create a default format definition (that is, without specifying the mode and state attributes) for all custom HTML types.

Displaying Server Side Errors

When a server side error occurs on submitting a form, SWE shows the same page again with the error message displayed within the page. For errors that occur outside of a form submission SWE continues to use the application's Error Page.

This error message display is mainly developed for showing error messages within a form. It is also used to show an error message in an error page to replace the to be depreciated pageitem.errorMsg way of showing error messages.

To display the error message within a form, place the following tags inside <swe:form> tag:

```
<swe:error>
   <swe:this property="FormattedHtml"/>
</swe:error>
```

The error messages are shown in plain text, but each error message takes a new paragraph. It is the responsibility of the enclosing HTML tags to modify the font and style of the error message. Sometimes, the error message may not be visible; this is because the font uses the same color as the background.

If the application developer does not use error tags in the swt files, the code automatically generates an error node (a CSSSWEErrorSWX instance). This automatically generated error node is inserted as the first child of the enclosing page/form node.

The syntax of the <swe:error> tag is as follows:

```
<swe:error>
   Usage:
   <swe:error property="FormattedHtml"/>
   or
   <swe:error>
      <swe:this property="FormattedHtml"/>
   <swe:error/>
```

This tag should be used within all <swe:form> tags.

You should also use this tag instead of the <swe:pageitem> tag mapped to the "_SWEErrMsg" item in the application's Error Page. The use of the "_SWEErrMsg" item is deprecated for 7.0.

An example of the use of this tag is:

```
<swe:form>
   <swe:error>
      <b><font color="red"> <swe:this property="FormattedHtml"/> </font></b>
   </swe:error>
```

```
...
</swe:form>
```

When the form is being rendered when there are no errors, the contents of the <swe:error> tag will be skipped.

Adding Graphics to Templates

To enhance the appearance or navigation of your application, you can create .GIF files and include links to them from the HTML pages.

Your Siebel Web installation includes three directories for your application. These directories will contain all the files used by your application, including the graphics files.

You should place your graphics in the public\lang\images directory. This directory gets created during the installation of the Siebel Web applications.

Adding Sorting Capabilities to you Application

To add sorting capabilities to your applications, you use the template file for the Sort Dialog by specifying the Application property Sort Web Page. You can use the Sort Dialog to show one or more instances of a list of fields to sort on, and the sorting order to use. You can invoke the Sort Dialog using a Control that invokes the SortOrder method. This control should be used only in base templates of List Applets.

You use the tag <swe:sort-field> to show the list of sortable fields and the sorting order options. This tag takes one attribute called sequence, which specifies the sort column order. This is a required attribute.

```
<swe:sort-field sequence=”1”/>
```

This tag will render two HTML select lists. The first select list will show the list of fields that can be sorted and have been mapped to <swe:control> tags in the base template for the applet. The second select list will show the two options for sorting order: Ascending and Descending. You can have as many <swe:sort-field> tags in the Sort Web Page as you want. Each <swe:sort-field> tag should specify the order in which the selected columns should be sorted, using the sequence attribute.

To create the link or button that would execute the sort, create a Web Page Item that invokes the ExecuteSort method. You do not have to specify the parameters View and Applet for this method; these will default to the currently active view and applet.

Example

Below is a fragment from a sample Sort Web Page.

```
<swe:form>

<table width=100% bgcolor=”#EEEEEE” border=0 cellspacing=0 cellpadding=3>
```

```
<tr>
   <td><swe:pageitem id="1" property="DisplayName"/></td>
   <!--"Sort By" Label -->
</tr>
<tr>
   <td><swe:sort-field sequence="1"/></td>
   <!-- First column to sort on -->
</tr>
<tr>
   <td><swe:pageitem id="2" property="DisplayName"/></td>
   <!-- "Then By" Label -->
</tr>
<tr>
   <td><swe:sort-field sequence="2" /></td>
   <!-- Second column to sort on -->
</tr>
<tr>
   <td><swe:pageitem id="2" property="DisplayName"/> </td>
   <!-- "Then By" Label -->
</tr>
<tr>
   <td><swe:sort-field sequence="3"/></td>
   <!-- Third column to sort on --></tr>
<tr>
   <td><swe:pageitem id="5" property="FormattedHtml"/></td>
   <!-- Execute Sort -->
</tr>
</table>
</swe:form>
```

About Cascading Style Sheets

The look and feel of user interface elements are controlled by cascading style sheets. Cascading style sheets contain classes that define elements such as color schemes and fonts. Cascading style sheet files (.css files) are located in:

- The Siebel Server installation directory

  siebsrvr_root\WEBMASTER\files*language_code*

- The Mobile or Dedicated Web Client installation directory

  client_root\PUBLIC*language_code*\FILES.

- The Tools installation directory

  tools_root\PUBLIC*language_code*\FILES

The following are examples of how you could use cascading style sheets to modify the look and feel of the user interface:

- Have text appear in the font of your choice
- Specify that size of text in points, pixels, and many other units
- Add any color or background color for images

The .SWT templates can be configured to use formatting tags. By storing style-related information in cascading style sheets rather than .SWT templates, you can increase the modularity and consistency of your applications and the ease with which the .SWT templates can be modified and reused.

Since style-related information stored in cascading style sheets is rendered slightly differently in different browsers, customers should test the results in both browsers unless their users are restricted to one or the other.

For more information about cascading style sheets, see *Siebel Developer's Reference*.

25 Configuring Keyboard Accelerators

Topics in This Chapter

About Accelerators

Keyboard accelerators are implemented using the command architecture. They are configured in Siebel Tools using the Accelerator object, which is a child of the Command object. Since accelerators are mapped directly to commands, the scope of the actions represented may be specific to the currently active applet, or it may apply to the application session as a whole. For example, an accelerator to initiate a new query will have specific focus on the current applet, while an accelerator to invoke the Site Map page is independent of the current application context.

Commands must be loaded into the active menu structure at run time in order to be available. This means that the command represented by each accelerator must be available to the user at a given point in time for the associated keyboard accelerator to be active. For a command to be available to the user, it must be associated with either the application menu or the applet-level menu for the currently active applet.

For more information about command objects, see "Creating Command Objects" on page 334.

Adding a New Keyboard Accelerator

You can add new keyboard accelerators.

To add an accelerator

1. Make sure the action to be performed by the accelerator is represented by a command object in the Siebel Repository.

   If not, you must add a command object. See "Creating Command Objects" on page 334 for information.

2. Make sure the command to be mapped to the new accelerator is included as part of the active menu hierarchy, at either the application or applet level, for the application contexts in which the accelerator will be active.

3. Navigate to the Commands object in the Object Explorer view.

4 Select the command that you want to modify.

5 Expand the Command object type, and then select the Accelerator child object.

6 Add a new record.

7 Specify the key sequence.

8 Specify the display name to be associated with the accelerator.

9 In the Platform field, specify the keyboard enablement mode or modes for which this accelerator will be active.

- Extended for extended mode only
- Basic for basic mode only
- All for both modes

10 Compile and check-in the project.

Keyboard accelerators for commands related to the Siebel Communications Server are configured through administrative screens in the application. They are not compiled in the Siebel Repository File using Siebel Tools. Any such accelerators defined through the Siebel Communications Server administrative screens will take precedence over accelerators defined in the Siebel Repository File for identical key sequences. For more information, see *Siebel Communications Server Administration Guide*.

Modifying Key Sequence

You can modify the key sequence for an existing accelerator.

To modify the Key Sequence for an accelerator

1 Navigate to the Commands object in the Object Explorer view.

2 Identify and highlight the command to modify.

3 Select the accelerator that you want to modify (Accelerator is a child object of the Command object).

4 Modify the Key Sequence property for the accelerator.

See "Guidelines for Configuring Keyboard Accelerators" on page 551.

5 Compile and check-in the project.

Hiding the Key Sequence

You can hide the key sequence so that it is not displayed in the user interface. The Key sequence for a given accelerator is defined by the Display Name property of the Accelerator Locale record. To hide the key sequence, leave this property blank.

Guidelines for Configuring Keyboard Accelerators

Key sequences that resemble the underlying command are useful since they help the user remember the accelerator (for example, "Ctrl+N" for "New Record"). However, it can be beneficial to keep several design considerations in mind to help maximize productivity gains when adding or modifying keyboard accelerators.

For applications running extended mode keyboard enablement, take care not to override native browser functionality that the user community uses regularly. For example, "Ctrl-C" copies a text string to the clipboard in Microsoft Internet Explorer 5.5. This could be a useful feature within the Siebel application browser environment for managing text.

Keep related accelerators grouped as much as possible in terms of mapped key sequences. For example, mapping key sequences that all start with "Ctrl+Alt…" for query management functions can assist users in remembering related accelerators.

Take care not to map frequently used commands to key sequences that are similar to those of commands with severe results. For example, mapping a frequently used command to "Ctrl+Alt+X" may lead to unwanted accidental logouts for users if the "Logout" command is mapped to "Ctrl+Shift+X".

Keyboard accelerators for commands related to the Siebel Communications Server are configured through administrative screens in the application. They are not compiled in the Siebel Repository File using Siebel Tools. Any such accelerators defined through the Siebel Communications Server administrative screens will take precedence over accelerators defined in the Siebel Repository File for identical key sequences.

For more information about Keyboard Accelerators, see *Applications Administration Guide*.

26 Configuring Spell Check

Topics in This Chapter

About Spell Check

Users can invoke Siebel Spell Check from an applet-level menu item. You configure this menu item by creating a "Check Spelling Field" user property for the applet where the Check Spelling button and the field to be checked are located.

Process for Configuring Spell Check

To configure spell check, perform the following tasks:

1. **Create a Spell Check button.** You need to create a Spell Check button for the applet containing the field to be checked. For information on performing this procedure, see "Creating a Spell Check Button."
2. **Set the Spell Check button user properties.** For information on performing this procedure, see "Defining Spell Check Button User Properties" on page 554.
3. **Edit the Web Applet template.** After creating a Spell Check button, you add it to the template where it will appear. For information on performing this procedure, see "Adding Spell Check Button to a Web Template" on page 555.
4. **Add the Spell Check business component to a business object**. Next, you add the Spell Check business component to the business object of the applet containing the field to be checked. For information on performing this procedure, see "Associating Spell Check Business Component to a Business Object" on page 555.
5. **Create a Spell Check menu item**. You need to create a Spell Check menu item for the applet containing the field you want to be checked. For information on performing this procedure, see "Creating a Spell Check Menu Item" on page 555.

NOTE: If you want to configure spell check for multiple fields on an applet, you must create a button for each field (tasks 1, 2, and 3).

Creating a Spell Check Button

To create a Spell Check button

1. In the Object Explorer, double-click the Applet object type to expand it.
2. In the Applets window, select the name of the applet for which you are creating a Spell Check button.
3. In the Object Explorer, select the Control object type and add a new record.

   Create the new record with the following values:

Field	Values for Non-Required Fields	Values for Required Fields
Name	ButtonCheckSpelling	ButtonCheckSpelling
Caption	Check Spelling	Check Spelling
Field	[field name]	*leave blank*
HTML Type	MiniButton	MiniButton
HTML Only	True	True
Method Invoked	ShowPopup	ShowPopup

NOTE: If the Method Invoked value does not appear in the picklist, type it in.

Defining Spell Check Button User Properties

To set the Spell Check button User Properties

1. In the Object Explorer, double-click the Controls object type to expand it.
2. Select the Control User Properties object type.
3. Click Control User Properties.
4. In the Control User Properties window, create new records for the following user properties, with the corresponding values:

Field	Value
Mode	Edit
Popup	Spell Checker Popup Applet
Popup Dimensions	560 X 350 (recommended initial size)

Adding Spell Check Button to a Web Template

To edit the Web Applet

1. In the Object Explorer, double-click the Applet object type to expand it.
2. In the Applets window, select the name of the applet for which you are creating a Spell Check button, and then right-click and choose Edit Web Layout.
3. In the Web Control toolbar, from the Mode drop-down list, select Edit.
4. In the Controls window, select the "Check Spelling" icon, and then drag it to a placeholder in the Web template.
5. In the Web template, right-click and choose Preview.

   You can see the Spell Check button as it will appear in the user interface.

Associating Spell Check Business Component to a Business Object

To add the Spell Check business component to a business object

1. In the Object Explorer, click the Business Object type.
2. In the Business Objects window, select the business object to which you want to add the Spell Check business component.
3. In the Object Explorer, double-click Business Object to expand it, and then select Business Object Component.
4. In the Business Object Component (child) window, add a new record.

   Create the new record with the following values:

Field	Value
BusComp	Spell Checker Applet VBC

Creating a Spell Check Menu Item

To create a Spell Check Menu Item

1. In the Object Explorer, click Applet, and then select the applet for which you want to create a Spell Check menu item.
2. Double-click the Applet object type to expand it, and then select Applet Method Menu Item.

3 In the Applet Method Menu Item window, add a new record.

Create the new record with the following values:

Field	Value
Command	Check Spelling
Menu Text	&Check Spelling
Position	2

4 If the field you are configuring for spell check is required, then do the following:

a Select the Applet User Property object type.

b In the Applet User Properties window, add a new record using following values:

Field	Value
Name	Check Spelling Field
Value	[Name of the control or list column mapped to the field that will use Spell Check]

27 Configuring the Customer Dashboard

Topics in This Chapter

About the Customer Dashboard

The customer dashboard provides access to key customer information, such as contact name and account number, in an area of the screen that remains persistent as the user navigates in the Siebel application. The customer dashboard is visible as a separate frame within the Siebel Web Client, below the screen tabs.

For more information about end user procedures, see *Fundamentals*.

How the Customer Dashboard is Populated With Data

The Customer Dashboard must be open to be populated with data. After the Customer Dashboard is populated, the information remains in the Customer Dashboard until the Clear Dashboard command is executed. During a session, all information populated in the Customer Dashboard is saved. The user can use Forward and Backward to display stored information. You can also configure a button on an applet that updates the Customer Dashboard with information from the currently selected row. For more information, see "Using Siebel VB and eScript to Populate the Customer Dashboard" on page 570.

The Customer Dashboard can be populated with data using any of the following methods:

- **Selected Record.** You can update the Customer Dashboard from a view by selecting a record and clicking the Update button in the Customer Dashboard. The Customer Dashboard then takes the primary business component for the view and updates the fields with data for this record.
- **Communications event.** When a user accepts an incoming call, the Customer Dashboard is automatically updated with contact information for the caller.
- **SmartScript answer.** You can configure the Customer Dashboard so that the answer to a question in a SmartScript automatically populates the Customer Dashboard.
- **Search Center results.** When the customer cannot be automatically identified from an inbound call, the user can search for the contact in Search Center and then click the Set Dashboard icon to populate the Customer Dashboard with the search results.

Architecture

The Customer Dashboard based on a virtual business component called Persistent Customer Dashboard, which is associated with the Persistent Customer Dashboard business object. The objects in the Siebel repository that are related to the Customer Dashboard are:

- **Persistent Customer Dashboard business object.** Groups together business components that can populate the Customer Dashboard with data.
- **Persistent Customer Dashboard business component.** Is a virtual business component.
- **Persistent Customer Dashboard business service.** Controls the functionality of the Customer Dashboard.
- **Persistent Customer Dashboard applet.** Displays data in the user interface.
- **Persistent Customer Dashboard view.** Displays applet in the user interface.

The method used for updating the Customer Dashboard is UpdateDashboard. If you want to configure a button to update the Customer Dashboard, use the InvokeMethod function and pass a set of name-value pairs such as:

- Source Name: 'Base View'
- BusComp Name: 'Contact'
- RowId: 'srowid'

See "Using Siebel VB and eScript to Populate the Customer Dashboard" on page 570 for more information.

Upon receiving the arguments, the methods evaluate the set of fields to be displayed, retrieve the data, and populate the Customer Dashboard.

Predefined Behavior

The Customer Dashboard has been configured to display a set of fields from several business components, including Account, Contact, Employee, Service Request, Asset Mgmt, and so on. You can configure the Customer Dashboard to display information from other business components as well. See "Process for Configuring the Customer Dashboard" on page 560 for more information.

You can see which business components the Customer Dashboard is configured to display by using Siebel Tools to review the list of Business Object Components associated with the Persistent Customer Dashboard business object. See "Adding Business Components to the Customer Dashboard Business Object" on page 560.

The Customer Dashboard can be populated with data from a single business component or multiple business components. However, the Customer Dashboard does not display data from multiple business components at one time. Rather, it is configured to display data in different contexts. For example, when the user is in the Accounts screen and clicks the Update button, account information is displayed; when the user is in the Contacts screen and clicks the Update button, contact information is displayed.

Enabling the Customer Dashboard

By default, the Customer Dashboard is enabled for Siebel Call Center, Siebel Sales, and Siebel Service. However, you can enable the Customer Dashboard for other applications as well.

To enable the Customer Dashboard

1. In Siebel Tools, navigate to the Persistent Customer Dashboard business service.
2. Verify that the Inactivate property is set to FALSE, which is the default setting.
3. Add the target application as a value for the Applications user property.

   For example, to activate Customer Dashboard for Siebel Employee Relationship Management, you would add Siebel ERM to the user property as shown in the table below:

[a]

Name	Value
Applications	Siebel Universal Agent; Siebel Field Service; Siebel Sales Enterprise; Siebel ERM

Process for Configuring the Customer Dashboard

You can configure the Customer Dashboard to display data from any business component.

To configure the Customer Dashboard to display data, perform the following tasks:

1. "Adding Business Components to the Customer Dashboard Business Object"
2. "Adding Business Component Lists User Properties to the Dashboard Business Service"
3. "Mapping Business Component Fields to the Dashboard Business Service" on page 561
4. "Creating Customer Dashboard Field Labels" on page 562
5. "Formatting Customer Dashboard Phone Number Fields" on page 563
6. "Configuring the Customer Dashboard GoTo View Drop-Down List" on page 564
7. "Configuring Labels for Customer Dashboard GoTo Views" on page 565

Adding Business Components to the Customer Dashboard Business Object

You may need to display data from a business component that the Customer Dashboard is not preconfigured to support. To do this, first you must add the business component to the Persistent Customer Dashboard business object.

This task is a step in the "Process for Configuring the Customer Dashboard" on page 560.

To add a new business component to the dashboard business object

1. In the Object List Editor, locate the Persistent Customer Dashboard business object.
2. Define a new Business Object Component (child of business object) for the business component.

Adding Business Component Lists User Properties to the Dashboard Business Service

User properties of the Persistent Customer Dashboard business service specify the business components and the list of fields available to display in the Customer Dashboard. These user properties are also known as Business Component Lists. Each user property name begins with List and is appended to make the name unique, for example List1, List2, and so on. The user property value identifies the name of a business component and the list of available fields.

For example, Table 69 shows the preconfigured user properties that identify the Contact and Opportunity business components and their corresponding fields.

Table 69. Example Persistent Customer Dashboard Business Service User Properties

Name	Value
List1	Contact;Last Name;First Name;Full Name;Email Address;Work Phone #;Account;Account Location;Fax Phone #;Job Title;Mobile Phone #
List2	Opportunity;Name;Account;Account Location;Oppty Id;Close Date;Sales Rep;Revenue;Sales Stage

This task is a step in the "Process for Configuring the Customer Dashboard" on page 560.

To add Business Component Lists user properties to the Dashboard business service

1. In the Object List Editor, locate the Persistent Customer Dashboard business service.
2. Add the List user property and define the business component name and field names as the user property's value.
   - **Name.** The syntax for the name is the word List followed by a number. For example, List1, List2, List3, and so on.
   - **Value.** The value of the user property lists the name of the business component and then the corresponding field names. Each value must be separated by a semicolon. See Table 69 for an example.

Mapping Business Component Fields to the Dashboard Business Service

You must map the available fields on the business component to fields on the Customer Dashboard applet. You do this by defining user properties for the Persistent Customer Dashboard business service.

The user property name identifies the Customer Dashboard fields, such as *Field 1, Field 2*, and so on.

The value of the user property defines the business component list and one of the available fields. The syntax for the value is the name of the list user property, for example *List1*, followed by the position of the field in the list for that user property. For example, *List 1.1* is the first field available from *List1, List1.2* is the second field available from *List1*, and so on. For more information about business component lists, see "Adding Business Component Lists User Properties to the Dashboard Business Service" on page 560.

For example, to display the Last Name field from the Contact business component (see List1 in Table 69 on page 561) in Field 1 of the Customer Dashboard, you would define a user property as shown in Table 70 below:

Table 70. Example User Property to Map Fields

Name	Value
Field 1	List1.1

You can display fields from more than one business component in a single Customer Dashboard field. To do this, you define multiple values for the Customer Dashboard field's user property. For example, suppose that when the Customer Dashboard is in the context of Contacts, you want Field 1 of the Customer Dashboard to display Last Name. However, when the Customer Dashboard is in the context of Opportunities, you want Field 1 to display Opportunity Name.

To map multiple business component fields to a single Customer Dashboard field, you define the user property as shown in Table 71. List 1.1. represents the first field of List one. List2.1 represents the first field of List2.

Table 71. Mapping Multiple Business Component Fields to a Single Dashboard Field

Name	Value
Field 1	List1.1;List2.1

The Customer Dashboard business service searches through the list of user properties, starting with Field, and looks for fields that are mapped to the Customer Dashboard from the current business component. For example, when the Contact business component is instantiated, the business service looks for Fields mapped from the Contact business component. Fields in the Customer Dashboard not mapped to fields in the current business component remain empty.

NOTE: The following fields are preconfigured for the Customer Dashboard, Field 1, Field 2, Field 3, Field 4, Field 5, Field 10, Field 12. Field 4 is formatted to display phone numbers.

This task is a step in the "Process for Configuring the Customer Dashboard" on page 560.

Creating Customer Dashboard Field Labels

The field labels that appear in the Customer Dashboard are dynamic; they change depending on the data being displayed in the Customer Dashboard. When no data is available for the Customer Dashboard, the labels for the default business component are displayed. The default business component is specified in the Customer Dashboard business service. Contacts has been preconfigured as the default business component for the Customer Dashboard.

The Siebel repository contains placeholder controls, such as Label 1, Label 2, and so on. There are also predefined business service user properties, also named Label 1, Label 2, and so on, that map these placeholder labels to fields on the Customer Dashboard.

If you add additional fields to the Customer Dashboard, you also need to define the labels that will replace placeholder labels at run time. You define the labels by creating an additional applet control for each business component field that you want to display. The naming convention for the applet control identifies it as a Label, and identifies the business component and field that determine when it should be displayed.

This task is a step in the "Process for Configuring the Customer Dashboard" on page 560.

To create an applet control to represent a label

1. Go to the Persistent Customer Dashboard applet.
2. Create a new applet control.
3. In the Name field, enter a name using the word Label, followed by a space, then followed by the business component name and field name separated by a dot.

   For example: Label ServiceRequest.SR Number
4. In the Caption field, specify the text that you want to appear in the Customer Dashboard.
5. Repeat this process for each label that you want to display in the Customer Dashboard.

Formatting Customer Dashboard Phone Number Fields

You can configure the Customer Dashboard to recognize different telephone extensions. You use a business service user property to define the parameters that associate your company's telephone switch extensions to their full-length phone numbers. The user property name is Phone Number Prefix. Three values, separated by semicolons, define the parameters. The values are as follows:

- The first value specifies the number of digits in an extension.
- The second value specifies the number of digits to remove from the front of the extension.
- The third value specifies the prefix to append to the beginning of the number.

Consider the example shown in Table 72:

Table 72. Phone Number Prefix Example

Name	Value
Phone Number Prefix 1	5;1;650555

The example shown in Table 72 would allow a user to dial the extension 24565. The extension has 5 digits. The first digit, 2, is removed. The prefix, 650555, is added. The resulting phone number is 650-555-4565.

Configuring the Customer Dashboard GoTo View Drop-Down List

The Customer Dashboard includes a drop-down list that allows users to navigate to additional views related to the current record. The list of views changes depending on the data currently displayed in the Customer Dashboard.

The list of views available in the Go To drop-down list are configured using business service user properties, such as View 1, View 2, and so on. At run time the Persistent Customer Dashboard business service searches through the list of user properties that start with View, finds the display name for the associated view, and then adds the name to the Go To drop-down list.

NOTE: The name of the view specified in the user property must exactly match the name as it is defined in the Siebel repository.

You can modify the views associated with the preconfigured View user properties or add additional views. The syntax for the user property is:

- **Name.** The word View followed by a number.
- **Value.** The value of the View user properties includes the following values separated by a semicolon:
  - Name of the business component
  - Name of the view
  - Name of the primary applet on the view
  - Name of the foreign key on a linked business component. This value is only necessary when you are navigating to a view based on a business component other than the current business component of the Customer Dashboard.

    For example, if the Customer Dashboard is configured to display data from the Contact business component, and the All Activities view is a view listed on the GoTo drop-down list, this value would specify the foreign key in the Action business component that points back to Contacts. It allows a query of all activities related to the contact currently displayed in the Customer Dashboard.

For example, Table 73 shows several preconfigured View user properties.

Table 73. Example View User Properties

Name	Value
View 1	Contact; All Activity List View; Activity List Applet With Navigation; Contact Id
View 2	Contact; Contact Activity Plan; Contact Form Applet
View 3	Contact; Agreement List View; Agreement List Applet No Parent; Contact Person Id

NOTE: View 1 shown in Table 73 specifies that when the Customer Dashboard is populated with data from the Contact business component, the All Activities view should appear in the GoTo View drop-down list. After the user selects the view from the drop-down list, only records for the current Contact ID appear in the view.

Configuring Labels for Customer Dashboard GoTo Views

You can specify the view labels for each view that you configure to appear in the GoTo drop-down list.

This task is a step in the "Process for Configuring the Customer Dashboard" on page 560.

To configure labels for GoTo views

1. Locate the Persistent Customer Dashboard applet.
2. Go to the Applet Controls.
3. Create a new control for the view that you have configured to appear on the GoTo drop-down list.

   The name of the control must be in the format of the word Label followed by the name of the View user property. For example, Label View 1 for View 1, and so on.
4. In the Caption property, enter the text you want to appear in the GoTo drop-down list.

Changing the Background Color and Border of the Customer Dashboard

You can change the background color and border color properties in the main.css file, which is located in PUBLIC*Language_Code*\FILES\ directory of your Siebel installation, where language code represents the three-letter code for your installation language pack.

To modify the background color or border

1. Locate the main.css file in the PUBLIC*Language_Code*\FILES\ directory of your Siebel installation.
2. Open the file with Notepad or other editor.
3. Find the following section and modify the values for dashbrdBorder and dashbrdBack as necessary.

```
/*---------------------*/

/*Dashboard Definitions*/

/*---------------------*/

.dashbrdBorder {background-color:#999999;}

.dashbrdBack {background-color:#f0f0f0;}
```

NOTE: in the example above, #999999 is medium grey and #f0f0f0 is light grey.

Changing the Size and Location of the Customer Dashboard

You can change the size and location of the Customer Dashboard. For example, you can make the Customer Dashboard appear in the bottom of the currently displayed view or you can make it occupy the complete horizontal space or a certain percentage of the content frame size.

NOTE: The Customer Dashboard is located outside of the Content Frame. You cannot move the Customer Dashboard inside of the Content Frame.

All changes can be done by modifying five template files:

- CCFrameContent_V.swt
- CCFrameContent_VD.swt
- CCFrameContent_VSD.swt
- CCFrameContent_VDT.swt
- CCFrameContent_VSDT.swt
- CCAppletDashboard.swt

A move to the left or right of the content frame will have to take into consideration the sizing issues present when Search Center is open. Moving the Customer Dashboard to the right side is not recommended because it breaks the connection between actions taken in Search Center and results returned in the main content area. When making changes to the Customer Dashboard location, be sure to test that the Customer Dashboard frame, content frame, and search center frame are working together properly.

Configuring Communication Events to Populate the Customer Dashboard

You can use communication events to populate the Customer Dashboard. Examples of communication events are inbound email message, voice call, or Web collaboration work item. The *Multichannel Def A* communication event has been preconfigured to populate the Customer Dashboard with contact information for certain communication events. However, you can configure any communication event to populate the Customer Dashboard for any business component, based on information passed to the event.

The API for the communication between communication events and the Customer Dashboard is a member function UpdatefromCTI of the Customer Dashboard business service. The CTI administration views are preconfigured to call InvokeMethod_ (with UpdateDashboard as a parameter) when a significant event occurs and passes variables, such as Phone number and Number of calls in queue, as arguments.

To populate the Customer Dashboard during a communications command or event, you need to call the method to update the Customer Dashboard and pass three parameters, including the business component, the field for that business component, and the value that you are getting from this communication.

For example, the parameters listed in Table 74 instruct the Customer Dashboard to populate with contact information for the contact whose Work Phone # matches the ANI of the inbound call.

Table 74. Customer Dashboard Parameters for Communications Events

Parameter	Example Value
ServiceMethod	Persistent Customer Dashboard.Update Dashboard from CTI
ServiceParam.Field	Work Phone #
ServiceParam.Value	{ANI}
ServiceParam.BusComp Name	Contact

You can also call the Customer Dashboard business service from the communications event log. See the steps below for an example.

1. Locate the Event Handler InboundCallReceived.
2. Click on the Associated Event Logs tab.
3. Drill down on the log LogIncomingCallContactFound.
4. In the log parameters, you define the following parameters:

Parameter	Example Value
ServiceMethod	Persistent Customer Dashboard.Update Dashboard from CTI
ServiceParam.Field	Id

Parameter	Example Value
ServiceParam.Value	{Contact.Id}
WorkTrackingObj.ContactId	{Contact.Id}

The Process of Configuring SmartScripts to Populate the Customer Dashboard

You can configure the Customer Dashboard so that the answer to a question in a SmartScript automatically populates the Customer Dashboard.

To configure SmartScripts to populate the Customer Dashboard, perform the following tasks:

1 "Activating the SmartScript Player" on page 568

2 "Mapping SmartScript Variables to Customer Dashboard Fields" on page 569

3 "Configuring SmartScripts to Save Answers" on page 569

NOTE: You cannot update the Customer Dashboard from VB or eScript that executes within a SmartScript. There is a one-to-one relationship between a user interface event and the ability to update a frame in the application. Because each user interface event within a SmartScript updates the SmartScript frame, it cannot also update the Customer Dashboard frame. If you were to pass parameters to the Customer Dashboard from VB or eScript within a SmartScript, the Customer Dashboard would receive the parameters but would not be able to display them.

Activating the SmartScript Player

When using the Customer Dashboard with SmartScripts you need to verify that the Notify Dashboard property of the Smart Script Player Applet (Tree Only) is set to true.

This task is a step in the "The Process of Configuring SmartScripts to Populate the Customer Dashboard" on page 568.

To verify that the Notify Dashboard user property is set to TRUE

1 Locate the applet named Smart Script Player Applet (Tree Only).

2 Locate the applet user property named Notify Dashboard.

3 Verify that the value of the user property is Y.

Mapping SmartScript Variables to Customer Dashboard Fields

You must map the variables in the SmartScript to fields on the Customer Dashboard. You do this by defining the SmartScript List user property of the Persistent Customer Dashboard business service. The mechanism for doing this is similar to defining user properties for a business component list. See "Mapping Business Component Fields to the Dashboard Business Service" on page 561.

The user property name is SmartScript List. The value for the user property specifies the variables from SmartScript answers that are to be displayed in the Configuration Dashboard.

This task is a step in the "The Process of Configuring SmartScripts to Populate the Customer Dashboard" on page 568.

To define the SmartScript List user property

1. Locate the Persistent Customer Service Dashboard business service.
2. Define a user property with the name SmartScript List and values that represent the variables from the SmartScript, as shown in the table below.

Name	Value
SmartScript List	Fname;Lname;Phone;Interests

NOTE: The values of the user property must exactly match the variable names specified in the SmartScript.

Configuring SmartScripts to Save Answers

To be able to pass SmartScript answers to the Customer Dashboard, you must configure the SmartScript to save the answers. You do this in the SmartScript Administration view.

This task is a step in the "The Process of Configuring SmartScripts to Populate the Customer Dashboard" on page 568.

To configure SmartScripts questions

1. Navigate to Site Map > SmartScript Administration > Questions.
2. In the Questions list, select a question.

3 In the More Info form, enter the name of the variable in the Save User Parameters field.

   This allows the answer to be saved as a global variable to the script.

   NOTE: The name of the variable must exactly match the name as it is listed in the SmartScript List user property of the Persistent Customer Dashboard business service. See "Mapping Business Component Fields to the Dashboard Business Service" on page 561.

4 From the Show drop-down list, choose Scripts.

5 In the Translation form, enter the name of the variables from each question.

   The syntax for entering the variables is the name of the variable enclosed in brackets, separated by spaces; for example, [Fname] [Lnmae] and so on.

6 Repeat the steps above for each question you need to configure for the Customer Dashboard.

   The values for the variables in the Dashboard Text field are passed to the Customer Dashboard when the SmartScript executes.

Using Siebel VB and eScript to Populate the Customer Dashboard

The Customer Dashboard provides APIs to pull information from or push information to the Customer Dashboard using Siebel VB or eScript. Since the Customer Dashboard is a separate frame in the application, it requires a UI event to update fields in the Customer Dashboard using Siebel VB or eScript. This means that you can only update the Customer Dashboard in this manner by adding a button to an applet and then calling the Update Dashboard command. When adding the button, make sure that you set the Target Frame View property to Dashboard.

NOTE: The Customer Dashboard architecture only allows one UI update for each user UI event. For example, if you put a button on a view, the clicking of the button is one UI event. For that event, you can only execute one UI update, such as updating the Customer Dashboard. The code behind a single button cannot have two UI updates, such as updating the Customer Dashboard and then going to a new view in the main frame of the application.

To add a button that calls the Update Dashboard Command

1 Locate the applet on which you want to place the button.

2 Add the button as an applet control and define the script for the control.

   See "About Customer Dashboard Commands" for a list of available commands.

3 Set the Target Frame property of the control to View to Dashboard.

About Customer Dashboard Commands

You can use Siebel VB or eScript to push information to the Customer Dashboard or pull information from the Customer Dashboard. Because the Customer Dashboard is a business service, you need to use the GetService ("Persistent Customer Dashboard") command.

There are two commands to pull information from the dashboard:

- "GetCurrentContactId". Gets the record Id for the current record populated in the dashboard.
- "GetDashboardFieldValue". Gets individual field values for fields currently populated in the dashboard.

GetCurrentContactId

This command returns the record Id for the current record populated in the Customer Dashboard. For example, if the record is from the Contact business component, then GetCurrentContactId returns the ContactId; if the record is from the Account business component, then GetCurrentContactId returns the AccountId.

No input argument should be specified.

The output argument is always "ContactId."

NOTE: The "ContactId" is a variable used by the Customer Dashboard, but this refers to the record ID for whichever business component is populated in the Customer Dashboard.

For example:

```
bs.InvokeMethod("GetCurrentContactId",inpargs,outargs); var fvalue =
outargs.GetProperty("Contact Id");
```

GetDashboardFieldValue

This command returns the current field value for the record populated in the Customer Dashboard. The input argument is the name-value pair for the Customer Dashboard field. The output argument is "Field Value."

For example:

```
inpargs.SetProperty("Field Name","Field 4");
bs.InvokeMethod("GetDashboardFieldValue",inpargs,outargs);  var fvalue =
outargs.GetProperty("Field Value");
```

Update Dashboard

This command is used to populate the Customer Dashboard with a new record.

- Source Name: Base View
- Buscomp Name: Contact
- RowId: E301

For example:

```
inpargs.SetProperty("Source Name","Base View", "Buscomp Name", "Contact", "RowId",
"E301"); bs.InvokeMethod("Update Dashboard",inpargs,outargs);No output argument
```

NOTE: In Siebel 7 versions 7.0.3 and 7.0.4 there are two spaces between "Buscomp" and "Name" in the second parameter. In subsequent versions, there is one space.

Example of Using Customer Dashboard Commands with Siebel eScript

The example below is a script, written in Siebel eScript, that uses the Customer Dashboard commands. It gets the contact ID, Field 4, and Field Time for the current record populated in the Customer Dashboard and prints those values to a file.

For more information about using Siebel VB, see *Siebel eScript Language Reference*.

```
function Script_Open ()
{
   var fn1=Clib.fopen("d:\\sabari5.txt", "wt");
   var bs = TheApplication().GetService("Persistent Customer  dashboard");
   var inpargs= TheApplication().NewPropertySet();
   var outargs = TheApplication().NewPropertySet();

   bs.InvokeMethod("GetCurrentContactId",inpargs,outargs);
   var fvalue = outargs.GetProperty("Contact Id");
   Clib.fprintf (fn1, "The current id in the dashboard = %s
\n",fvalue);

   inpargs.SetProperty("Field Name","Field 4");
   bs.InvokeMethod("GetDashboardFieldValue",inpargs,outargs);
   var fvalue = outargs.GetProperty("Field Value");
   Clib.fprintf (fn1, "The Account Name in the dashboard =  %s  \n",fvalue);

   inpargs.SetProperty("Field Name","Field Time");
   bs.InvokeMethod("GetDashboardFieldValue",inpargs,outargs);
   var fvalue = outargs.GetProperty("Field Value");
   Clib.fprintf (fn1, "The current time of the agent/customer in the dashboard =  %s
\n",fvalue);

   Clib.fclose(fn1);
return(ContinueOperation);
}
```

Example of Using Customer Dashboard Commands with Siebel VB

Below is an example script written in Siebel VB that uses the Customer Dashboard commands. It gets the contact ID, Field 4, and Field Time for the current record populated in the Customer Dashboard and prints those values to a file.

For more information about using Siebel eScript, see *Siebel VB Language Reference*.

```
Sub Script_Open

     Dim bs as Service
     Dim inpargs as PropertySet
     Dim outargs as PropertySet
     Dim fvalue as String
```

```
Open "d:\sabari.txt" for Output as #1
Set bs = TheApplication().GetService("Persistent Customer  dashboard")
     Set inpargs = TheApplication.NewPropertySet
     Set outargs  = TheApplication.NewPropertySet

     bs.InvokeMethod "GetCurrentContactId",inpargs,outargs
     fvalue = outargs.GetProperty("Contact Id")
     Write #1, "The current id in the dashboard = " & fvalue

     Inpargs.SetProperty "Field Name","Field 4"
     bs.InvokeMethod "GetDashboardFieldValue",inpargs,outargs
     fvalue = outargs.GetProperty("Field Value")
     Write #1," The Account Name in the dashboard =  "& fvalue
     Close #1

End Sub
```

About Dual Personalization

The Personalization engine has the ability to personalize the Call Center application based on both the agent's profile and the customer's profile. The agent's profile is loaded when the agent logs into the Call Center application. The customer's profile is loaded when the customer information is populated in the Customer Dashboard. This allows the agent to see customer-specific information based on the personalization rules created by your Siebel administrator.

For example, based on the customer's profile you could show a different applet or view to the agent. You could have a Recommended Products applet which only shows products for this customer based on products he previously purchased.

To access the profile information you create personalization rules. The Me.attribute allows you to access the agent's profile information and the You.attribute to access the customer's profile information. Examples of these commands are displayed below.

- GetProfileAttr("You.Last Name");
- GetProfileAttr("Me.Last Name");

For more information about profile attributes and creating personalization rules, see *Personalization Administration Guide*.

28 Online Help Development

Topics in This Chapter

NOTE: The topics in this section assume that you are familiar with Siebel Tools help development, HTML authoring, the use of cascading style sheets, and JavaScript.

Online Help Implementation Overview

The topics in this section provide a high-level overview of the implementation of online help in Siebel 7 employee, partner, and customer applications. It also provides some tips about editing the HTML files that make up the help system.

The following topics are included:

Employee Applications

Employee applications are generally used by internal employees of an enterprise. Siebel Call Center and Siebel Sales are examples of employee applications. In these applications, the help system is delivered in HTML format, and the help is context-sensitive at the screen level. When a user accesses help, the application calls the Siebel Web Engine (SWE) GotoPage method, which uses SWE code to display the correct help topic in a separate browser window.

The start page of the employee applications help system is siebstarthelp.htm, shown in Figure 135. For information about using the help system, see "Using the Employee Applications Online Help" on page 576.

"Help Source Files" contains information about the files that make up the employee applications help system.

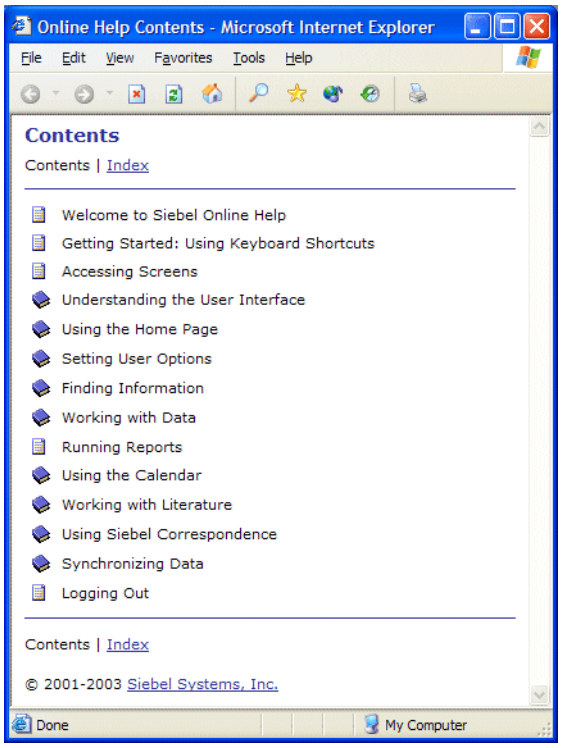


Figure 135. Start Page Employee Applications Help

Using the Employee Applications Online Help

After you have accessed the online help, you can use the Web browser's functionality to navigate in the help system and to print topics. In addition, the online help has an index that lets you find topic by keyword.

To link to help topics you recently visited

- To return to the last topic you viewed, click the Web browser's Back button.
- To view a topic you viewed before clicking the Web browser's Back button, click the Web browser's Forward button.

To print a help topic

- Click the Web browser's Print button.

  The current HTML page is printed to your default printer. The HTML page may consist of more than one topic.

To return to the main contents page

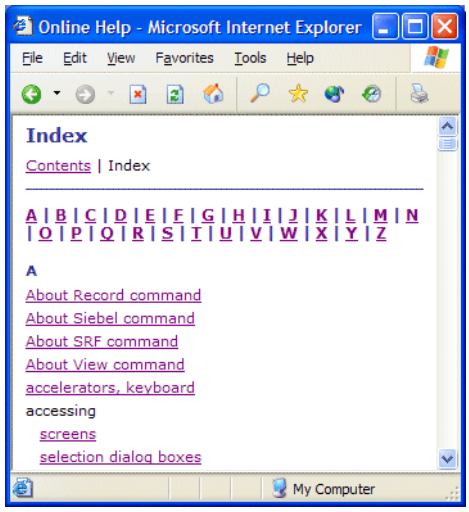
- In the help window, click the Contents hyperlink that appears at the start and end of each HTML page that makes up the help system.

  The start page of the employee applications help system appears.

To search the help by keyword

1. In the help window, click the Index hyperlink.

   The Index topic, shown below, appears.



2. Click a letter hyperlink at the start of the page to go to entries for that letter.
3. Click the hyperlink for the keyword.

   The help topic appears.

Customer Applications

Customer applications are generally used by external partners, customers, and prospects of an enterprise. Siebel eSales and Siebel eService are examples of customer applications. In these applications, the help system is delivered in HTML format, and the applications are configured to show the start page of the help system in a separate browser window whenever a user accesses help. This is done by using the SWE GotoURL method.

The start page of the customer applications help system is siebcomgeneric.htm, shown in Figure 136. For information about using the help system, see "Using the Customer Applications Online Help" on page 578.

"Help Source Files" contains information about the files that make up the customer applications help system.

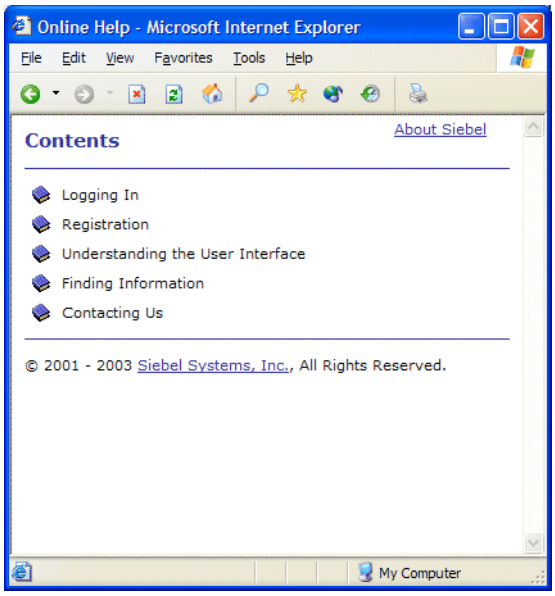


Figure 136. Start Page Customer Applications Help

Using the Customer Applications Online Help

Once you have accessed the online help, you can use the Web browser's functionality to navigate in the help system and to print topics.

To link to help topics you recently visited

- To return to the last topic you viewed, click the Web browser's Back button.
- To view a topic you viewed before clicking the Web browser's Back button, click the Web browser's Forward button.

To print a help topic

- Click the Web browser's Print button.

  The current HTML page is printed to your default printer. The HTML page may consist of more than one topic.

To return to the main contents page

- In the help window, click the Contents hyperlink that appears at the start and end of each HTML page that makes up the help system.

  The start page of the customer applications help system appears.

About Editing HTML Files

Siebel applications help files reside in a folder called "help." When you have determined which HTML files you need to change, it is recommended that you copy those files to your local machine to make changes.

Testing and Distributing Changes

After you make your changes, you should verify your changes before distribution.

To test your changes

1. Use a standard HTML authoring tool to verify links.

   NOTE: You should verify the links in the changed HTML file, and also make sure no links were broken in the existing HTML files.

2. In a Web browser, open the HTML file you changed and review the content.

When you have completed testing, you must distribute the updated files to the appropriate Siebel Servers, Siebel Mobile Web Clients, or Siebel Dedicated Web Clients.

Employee Applications

The topics in this section describe the details of the help implementation in Siebel 7 employee applications. It explains the location of the help files and provides information about using Siebel Tools to change help calls. In addition, different options for customizing the online help to suit your requirements are discussed. If you customized the online help in earlier versions of your Siebel application, you can also find instructions for migrating your customized online help.

The following topics are included:

NOTE: The help implementation is identical for employee and partner applications. Therefore, the information contained in the topics above applies to both employee and partner applications.

Location of Employee Application Help Files

The location of the help files is determined by the location where you installed your Siebel applications and by the type of Siebel client:

Siebel Web Client

Siebel Web Client runs in a standard browser from the client personal computer and does not require any additional persistent software installed on the client. The browser connects through a Web server to the Siebel eBusiness Application server, which executes business logic and accesses data from the Siebel Database.

In this implementation, help files are installed in the following location on the server:

<*install dir*>\public\<*install language*>\help, where

- *install dir* is the directory where you installed the Siebel Web Server Extensions
- *install language* is the language you selected during installation

During the installation process, your Web server is configured so that
<*install dir*>\public\<*install language*>\
becomes the root directory for the URL
http://<*hostname*>/<*Siebel application name*>

When a Help Identifier property in a Screen object maps to a URL in the Help Id object, that URL is relative to http://<*hostname*>/<*Siebel application name*>.

Example

If you are running Siebel Call Center on the server siebsrvr, when you request the help for the Accounts screen, the page that will appear is
http://siebsrvr/callcenter/help/siebaccounts.htm. This means that you can put the help files in any directory that can be referenced from
http://<*hostname*>/<*Siebel_application_name*>.

Assume that in <*install dir*>\public\<*install language*>\, you create a directory called customizedhelp and you create a new help topic file for the Accounts screen, called accountshelp.htm. After that, you configure your Web server so that this directory is exposed to a Siebel application user. In this case, you would create Help Id objects that point to the URL customizedhelp; the Help Id object properties would be:

Property	Value
Name	ID_SCREEN_ACCOUNTS
Project	Repository Help Id
Type	Screen
HTML Help URL	customizedhelp/ accountshelp.htm

Siebel Dedicated Web Client and Siebel Mobile Web Client

Siebel Dedicated Web Client is a Microsoft Windows client delivered through a Web browser that provides direct connectivity to a database server. It requires software to be installed on the client machine, but does not require a local database, Web server, or Siebel eBusiness Application server for serving up interactive user sessions. Siebel Server is still required for functionality like Assignment Manager and Replication.

Siebel Mobile Web Client is a portable Microsoft Windows client delivered through a Web browser that is designed for local data access and does not need to be connected to a server. Siebel Mobile Web Client meets the needs of field professionals who do not have continuous access to a network. Siebel Mobile Web Client uses a local database on each mobile machine. Periodically, the client must access the Siebel Remote Server through a dial-up, WAN, or LAN connection to synchronize data changes with the Siebel Database on the database server and Siebel File System. This client requires installation of Siebel software on the user's personal computer.

The software installed on the user's machine for the Siebel Dedicated Web Client and Siebel Mobile Web Client is identical—the only difference is the type of connectivity provided.

In these implementations, the location of the help files is determined by the installation directory on the client:

<*install path*>\public\<*install language*>\help, where

- *install path* is the complete path to the location where you installed the Siebel application
- *install language* is the language you selected during installation

For example, D:\sea\webclient\public\enu\help.

During the installation process, your local Web server is configured so that
<*install path*>\public\<*install language*>\
becomes the root directory for the URL http://localhost/.

When a Help Identifier property in a Screen object maps to a URL in the Help Id object, that URL is relative to http://localhost.

Example

If you are running Siebel Call Center on your local machine, when you request the help for the Accounts screen, the page that will appear is
http://localhost/help/siebaccounts.htm. This means that you can put the help files in any directory that can be referenced from http://localhost/.

Assume that in <*install path*>\public\<*install language*>\, you created a directory called customizedhelp and you created a new help topic file for the Accounts screen, called accountshelp.htm.

In this case, you would create Help Id objects that point to the URL customizedhelp; the Help Id object properties would be:

Property	Value
Name	ID_SCREEN_ACCOUNTS
Project	Repository Help Id
Type	Screen
HTML Help URL	customizedhelp/ accountshelp.htm

Online Help and Siebel Tools

In Siebel Tools, Screen, View, and Help Id objects are used to establish the link between a screen or a view and a help topic file.

NOTE: Standard Siebel eBusiness Applications do not contain view-level help topic references; they are all at the screen level. See "Implementing Help for a View" on page 586 for instructions about adding view-level help references.

The following topics are included:

Screen and View Objects

Each Screen and View object has a Help Identifier property that is used to establish the link with the Help Id object.

The format of the Help Identifier property is ID_*type_objdefname*, where:

- *type* is SCREEN or VIEW
- *objdefname* identifies the screen or view

Help Id Objects

Each Help Id object has a HTML Help URL property that is used to identify the HTML file that contains the help topics for the screen or the view by mapping the Help Identifier to a specific URL. You can use the same value for the HTML Help URL property for different Help Id objects.

The format of the HTML Help URL property is *help*/*helptopics*.htm, where:

- *help* is the server folder where the HTML topic files reside
- *helptopics*.htm is the HTML file that appears when a user invokes help

See "Location of Employee Application Help Files" on page 579 for additional information about where the help files are installed.

Help Properties of Screens and Views

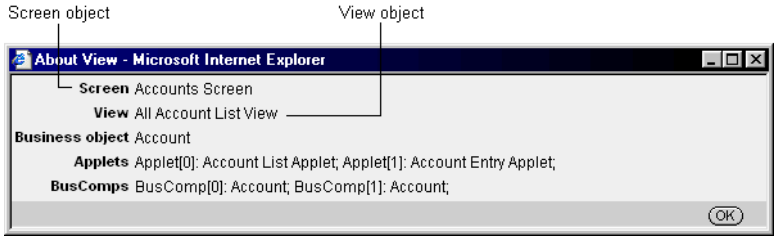
To determine which HTML file contains the help topics for a screen or a view you must: use your Siebel application to determine the Screen or View object used in the screen or view, then use Siebel Tools to find the Help Identifier property of the Screen or View object and then find the corresponding HTML Help URL property in the Help Id object.

NOTE: Some screens are associated with generic help topics. In this case, you must use a different filename if you want to customize the help. See "Customizing Help for Screens with Generic Help Topic Files" on page 588 for instructions.

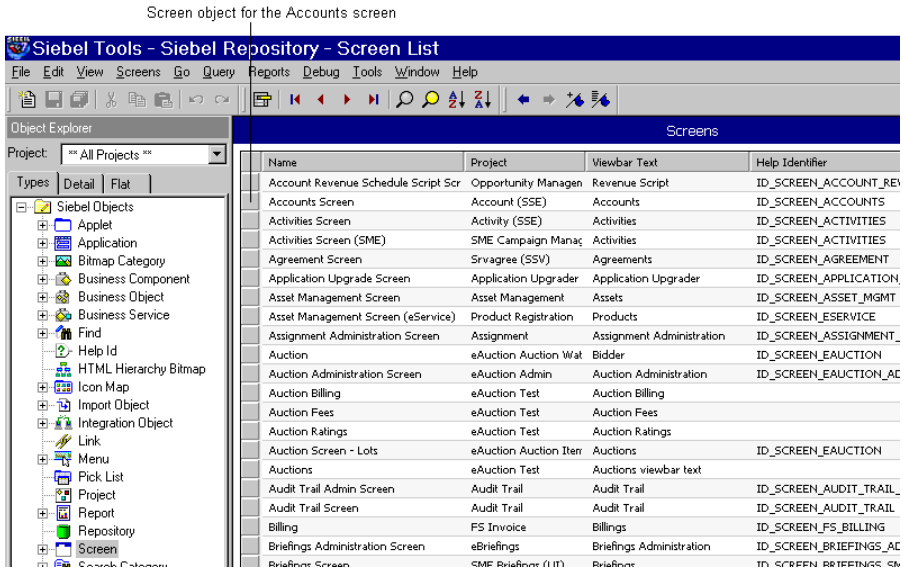
Example

To determine which HTML file contains the help topics for the Accounts screen, you must find which Screen object is used in the Accounts screen.

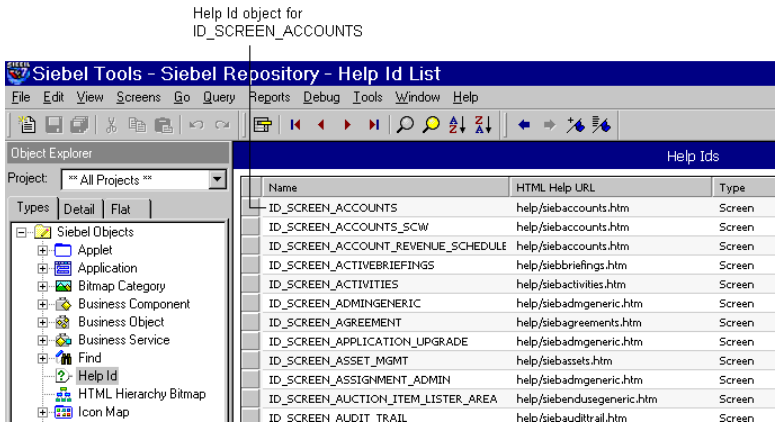
To do this, in your Siebel eBusiness Application, navigate to the Accounts screen, then choose Help > About View to access the About View dialog box, shown below.



In Siebel Tools, find the Accounts screen object and find the value of its Help Identifier property (ID_SCREEN_ACCOUNTS) in the Object List Editor window, shown below.



When you know the value of the Help Identifier property of the Screen object (ID_SCREEN_ACCOUNTS), you can find the value of the Help Id object's HTML Help URL property (help/siebaccounts.htm) in the Object List Editor window, as shown below.



Default Help Topic

The start page of the employee applications help system (siebstarthelp.htm) is defined as the default help topic for employee applications. When a user accesses help from a screen or view for which the Help Identifier property is not defined, siebstarthelp.htm appears in a separate browser window.

To change the default help topic

1. In Siebel Tools, in the Object Explorer (Types tab) find and select the Web Page object type.
2. In the Object List Editor, find and select the CC Help Page object definition.
3. In the Object Explorer (Types tab), expand the Web Page object type and select the Web Page Item object type.
4. In the Object List Editor, modify the Caption property of the Online Help object definition to point to your default help topic.

   For example, change the Caption property from `help/siebstarthelp.htm` to `help/index.htm`.
5. Recompile the repository file.
6. Test your changes and distribute the repository file and your new default topic file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   For example, for Siebel Web Client implementations, copy index.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

   For more information, see "Testing and Distributing Changes" on page 579.

Implementing Help for a Screen

If you add a custom screen in your implementation of Siebel eBusiness Applications, you can create help for that screen. To do this, you must use Siebel Tools to define the Screen object with a Help Identifier property, and then add a Help Id object with a HTML Help URL property for the screen. The documentation team can develop the content using the name of the file that will be distributed to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

To add help for a new screen

1. In Siebel Tools, if the Screen object does not exist, create it.

   For example, create the My New Screen Screen object.

   For more information, see "Process of Creating Screens and Screen Views" on page 304.
2. In the Screen object, define the Help Identifier property.

   For example, ID_SCREEN_MYNEWSCREEN.
3. Create a new Help Id object with the following properties:

Property	Value	Example
Name	Value of the Help Identifier property of the screen	ID_SCREEN_MYNEWSCREEN
Project	Repository Help Id	Repository Help Id
Type	Screen	Screen
HTML Help URL	Name of the file that will contain the help content	help/mynewscreen.htm

4 Recompile the repository file.

NOTE: You must compile the Repository Help Id project *and* the Screen object. In our example, you must compile the My New Screen Screen object and the Repository Help Id project.

5 Create a new HTML file with help content for the screen.

6 Save the HTML file using the name defined in the HTML HELP URL property of the Help Id object associated with the screen.

For example, mynewscreen.htm.

7 Test your changes and distribute the repository file and the new HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

For example, for Siebel Web Client implementations, copy mynewscreen.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

For more information, see "Testing and Distributing Changes" on page 579.

Implementing Help for a View

To add help for a view, you must use Siebel Tools to define the View object with a Help Identifier property, and then add a Help Id object with a HTML Help URL property for the view. At the same time, the documentation team can develop the content, and they can use the name of the file that will be distributed to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

To add help for a view

1 In Siebel Tools, if the View object does not exist, create it.

For more information, see "Process of Creating Views" on page 285.

2 In the View object, define the Help Identifier property.

For example, for the Opportunity Detail - Contacts View, use ID_VIEW_OPPORTUNITY_DETAIL_CONTACTS as the Help Identifier.

NOTE: If you leave the Help Identifier property blank for the View object, the Screen-specific help will be used as the default help topic for the view.

3 Create a new Help Id object with the following properties:

Property	Value	Example
Name	Value of the Help Identifier property of the view	ID_VIEW_OPPORTUNITY_DETAIL_CONTACTS
Project	Repository Help Id	Repository Help Id

Property	Value	Example
Type	View	View
HTML Help URL	Name of the file that contains the help topics	help/ siebopportunities_detailcontacts.htm

4 Recompile the repository file.

NOTE: You must compile the Repository Help Id project *and* the View object. In our example, you must compile the Opportunity Detail - Contacts View object and the Repository Help Id project.

5 Create a new HTML file with the help content for the view.

6 Save the HTML file using the name defined in the HTML HELP URL property of the Help Id object associated with the view.

For example, siebopportunities_detailcontacts.htm.

7 Test your changes and distribute the repository file and the new HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

For example, for Siebel Web Client implementations, copy siebopportunities_detailcontacts.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

For more information, see "Testing and Distributing Changes" on page 579.

Changing the Keyboard Shortcut for Accessing Help

Since most users are accustomed to accessing help by pressing the F1 key, you may want to map F1 to display online help in your Siebel application. To do this, you must add an accelerator in Siebel Tools.

To change the keyboard shortcut for accessing help

1 In Siebel Tools, select the Command object type in the Object Explorer.

The Commands list appears in the Object List Editor.

2 In the Commands list in the Object List Editor, find the Contents Help (SWE) command object and select the row.

3 In the Object Explorer, click Accelerator.

The Accelerators list appears below the Commands list in the Object List Editor.

4 Change the two Accelerator objects as shown below:

Name	Display Name	Key Sequence
1	F1	F1
2	F1	F1

5 Recompile the repository file.

6 Distribute the repository file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

Help Menu Items

If you create a new application using Siebel Tools, you can add options to the Help menu in your application. For example, you may want to add an option that allows the user to access the *Siebel Bookshelf* from within the application.

The Help application menu option is configured using the Command and Menu objects.

The Command object created for online help is Contents Help (SWE).

For instructions about using the Command and Menu objects, see "Configuring Toolbars and Menus" on page 327.

Customizing and Adding Help

You can customize help content included for a screen to suit your Siebel implementation and requirements. Standard Siebel eBusiness Applications do not include help files for views. If you want to add help at the view level, you must implement help for the view and then create a new HTML file with the help content for the view. See "Implementing Help for a View" on page 586 for instructions.

The following topics are included:

Customizing Help for Screens with Generic Help Topic Files

Some screens are associated with generic help topics, which means that the HTML Help URL property of the Help Id object for a screen is one of the following:

- help/siebadmgeneric.htm for administrative screens

- help/siebendusegeneric.htm for end-user screens

See "Help Properties of Screens and Views" on page 583 for information about finding the HTML Help URL property.

To customize help for screens with generic help content

1. Open the HTML file referenced in the HTML Help URL property (siebendusegeneric.htm or siebadmgeneric.htm).
2. If the following <META> tag appears at the top of the file, remove it.

   ```
   <META HTTP-EQUIV="REFRESH" CONTENT="1; URL=siebwelcome.htm">
   ```

3. Save the file with a different name.
4. In Siebel Tools, update the HTML Help URL property of the Help Id object to reflect the new filename.
5. Follow the instructions in "Customizing Help Content" on page 589.

Customizing Help Content

Since the help consists of HTML pages, you can use any HTML editor to change the content of a help topic.

To customize help content for a screen

1. Find the HTML Help URL property associated with the screen.

   See "Help Properties of Screens and Views" on page 583 for instructions.

   For example, the HTML Help URL property for the Opportunities screen is help/siebopportunities.htm.

   NOTE: If the HTML Help URL property for the screen is siebendusegeneric.htm, siebadmgeneric.htm, or siebstarthelp.htm, use the instructions in "Customizing Help for Screens with Generic Help Topic Files" on page 588 instead.

2. Open the HTML file, make your changes, and save the file.

   For example, open siebopportunities.htm, make your changes, and save the file.

   NOTE: If a redirect tag exists in the file, be sure to remove the tag. The following is an example of a redirect tag:
   `<META HTTP-EQUIV="REFRESH" CONTENT="1; URL=siebwelcome.htm">`

3. Test your changes and distribute the updated HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   For example, for Siebel Web Client implementations, copy siebopportunities.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

   For more information, see "Testing and Distributing Changes" on page 579.

Adding Help for a Screen

If you added a custom screen to your implementation of Siebel eBusiness Applications, you can add help for that screen. See "Implementing Help for a Screen" on page 585 for instructions.

Adding Help for a View

To add help for a view, you must name the HTML topic file using the HTML Help URL property of the view. See "Implementing Help for a View" on page 586 for instructions.

Customizing the Help Index

The index topic in the help system allows the user to find topics by keyword (see "To search the help by keyword" on page 577). The index topic is a set of links to topics based on keywords. When you customize the content of the help system, you may want to update the entries in the index to reflect the new content. Another option is to remove access to the index topic entirely.

To customize the help index

1. Open siebindex.htm, make your changes, and save the file.
2. Test your changes and distribute the updated siebindex.htm to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   For example, for Siebel Web Client implementations, copy the updated siebindex.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

   For more information, see "Testing and Distributing Changes" on page 579.

To remove access to the index topic

1. Open one of the HTML files, except siebindex.htm, that is part of the help system.
2. Delete the statements that reference siebindex.htm and save your changes.
3. Repeat Step 1 and Step 2 for each HTML file in the help system, except siebindex.htm.
4. Test your changes and distribute all the updated HTML files to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   For example, for Siebel Web Client implementations, copy all the updated HTML files to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

   For more information, see "Testing and Distributing Changes" on page 579.

Migrating Help

If you customized the online help in Siebel 98 (Version 4), Siebel 99 (Version 5), or Siebel 2000 (Version 6) and the customization was mostly related to task topics, it may be more effective to rewrite the content because the navigation in Siebel eBusiness Applications has changed considerably.

The following topics are included:

Help Migration Options

There are several ways to migrate customized help content to HTML format:

- Update the Siebel HTML topic file with information from your customized topic file from a previous release.

  See "Updating Siebel Topic Files with Custom Content" on page 592.

- Convert existing content to HTML using the filename found in Siebel Tools.

  See "Converting Content to HTML Format Using Siebel File Names" on page 593.

- Convert existing content to HTML using a filename of your choice, and update the HTML Help URL property.

  See "Converting Content to HTML Format Using Custom File Names" on page 594.

If all users will be running on a MS Windows platform, you can use your current compiled Microsoft Windows help system as the help system for your Siebel application. See "Using WinHelp" on page 595 for more information.

NOTE: In this section, *rich text editor* refers to a text editor that supports hidden text and footnotes, such as Microsoft Word.

Sample Scenario

For clarity purposes, the procedures in this section are illustrated by an example that assumes that you customized the help content for the Accounts screen to suit your implementation. In earlier versions of Siebel applications, you would have made these changes in the sa_acct.rtf source topic file.

Updating Siebel Topic Files with Custom Content

You can update the online help with customized information contained in your help system from a previous Siebel release using the Siebel topic files included with your Siebel eBusiness Applications.

Advantages

- Maintains formatting, layout, and navigation elements of the original, Siebel-delivered online help.
- Links to the cascading style sheet remain in place.

Disadvantages

- None

To update the Siebel HTML file with information from a customized .rtf file

1. Find the help properties of the screen.

   See "Help Properties of Screens and Views" on page 583 for instructions.

   In the sample scenario, the HTML Help URL property associated with the Accounts screen is help/siebaccounts.htm.

2. Open the HTML file referenced in the HTML Help URL property.

   In the sample scenario, open siebaccounts.htm.

   NOTE: If a redirect tag exists in the file, be sure to remove the tag. The following is an example of a redirect tag:

   ```
   <META HTTP-EQUIV="REFRESH" CONTENT="1; URL=siebwelcome.htm">
   ```

3. Open the source .rtf file (used to create the help content in the previous version) in a rich text editor.

   In the sample scenario, open sa_acct.rtf.

4. Use copy and paste functionality to copy content from the .rtf file to the HTML file.

   In the sample scenario, copy content from sa_acct.rtf to siebaccounts.htm.

5. Apply the appropriate HTML tags to format content and save the HTML file.

   See "Help Source Files" for a description of the cascading style sheet used in the help.

   In the sample scenario, save siebaccounts.htm.

6. Test your changes and distribute the updated HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   In the sample scenario, for Siebel Web Client implementations, replace *<install dir>*\public*<install language>*\help\siebaccounts.htm with your version of siebaccounts.htm created from the .rtf file.

   For more information, see "Testing and Distributing Changes" on page 579.

Converting Content to HTML Format Using Siebel File Names

If you customized the help content in previous versions of your Siebel application, you can use the method described in this section to convert the customized help content to HTML format. Use your source topic file (rich text format, .rtf) as a starting point and follow the instructions.

Advantages

- The new help content exactly matches the customized content of the previous release.

Disadvantages

- Creates a risk of unexpected formatting results if the person customizing the help is not familiar with HTML.
- Creates risk of losing navigation elements in the help system (to table of contents and index topics) if the correct HTML code is not inserted in the new HTML file.

To convert a customized .rtf file to HTML using the Siebel filename

1. Find the help properties of the screen.

   See "Help Properties of Screens and Views" on page 583 for instructions.

   In the sample scenario, the HTML Help URL property associated with the Accounts screen is help/siebaccounts.htm. The Siebel filename for the topic file is "siebaccounts.htm."

2. Open the source .rtf file (used to create the help content in the previous version) in a rich text editor.

   In the sample scenario, open sa_acct.rtf.

3. Save the file in HTML format, using the appropriate Siebel filename.

   In the sample scenario, save sa_acct.rtf as siebaccounts.htm.

4. Open the newly created HTML file.

5. In the HTML file, add a reference to the Siebel help cascading style sheet (siebhelp.css) and add the necessary blocks of code to implement the navigation hyperlinks.

   NOTE: You can copy this information from one of the Siebel-delivered HTML files.

6. Clean up the HTML code to use styles defined in the style sheet and save the HTML file.

7. Test your changes and distribute the updated HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   In the sample scenario, for Siebel Web Client implementations, replace <*install dir*>\public\<*install language*>\help\siebaccounts.htm with your version of siebaccounts.htm.

   For more information, see "Testing and Distributing Changes" on page 579.

Converting Content to HTML Format Using Custom File Names

You can use existing content in .rtf format and convert it to HTML using your own file naming convention.

Advantages

- Requires very little work from your help developer or technical writer.

Disadvantages

- Siebel Tools developers have to update the help properties for each topic file that does not use a Siebel name.
- Increases risk of errors because the names of files may be entered incorrectly in Siebel Tools.

To convert existing content to HTML using a filename of your choice

1. Open the source .rtf file (used to create the help content in the previous version) in a rich text editor.

   In the sample scenario, open sa_acct.rtf.

2. Save the .rtf file in HTML format, with a name you choose.

   In the sample scenario, save sa_acct.rtf as sa_acct.htm.

3. Test your changes by opening the HTML file in a Web browser.
4. In Siebel Tools, find the help properties of the screen.

   See "Help Properties of Screens and Views" on page 583 for instructions.

5. Update the HTML Help URL property of the Help Id object to reflect the correct filename.

   In the sample scenario, update the HTML Help URL property for the ID_SCREEN_ACCOUNTS Help ID object to be help/sa_acct.htm.

6. Do one of the following:

   - If you only changed HTML Help URL properties, recompile the Repository Help Id project to create an SRF and implement the changes.
   - If you made other changes, recompile all affected projects to create an SRF and implement the changes.

7. Test your changes and distribute the repository file and the new HTML file to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   In the sample scenario, for Siebel Web Client implementations, copy sa_acct.htm to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

   For more information, see "Testing and Distributing Changes" on page 579.

Using WinHelp

If all users will be running on a Microsoft Windows platform, you can use your current compiled Microsoft Windows help system (WinHelp) as the help system for your Siebel eBusiness Applications.

It is important to note that this solution is *not recommended* and that there are several drawbacks to this implementation:

- Each time a user invokes help, the Web browser's File Download dialog box will appear, and the user must respond to access the help. The only way to avoid this is for your administrator to change security settings.
- When a user invokes help at a screen, the default topic in the WinHelp file appears in the help window, not the context-sensitive topic associated with the screen.

  In WinHelp, you specify the default topic for the help file in the [OPTIONS] section of the help project file (.hpj). If you do not specify a default topic, WinHelp uses the first topic of the first file listed in the help project (.hpj) file.
- From the help window, the user can access the Index, but the table of contents (usually available through the Contents tab of the Help Topics window) is not available. Microsoft is aware of this problem, but since WinHelp is no longer the Microsoft method of choice for help delivery, Microsoft will not fix this defect.

NOTE: The following procedure assumes that you want to use siebhelp.hlp (a WinHelp file) as the help system for your Siebel eBusiness Applications.

To use a compiled WinHelp file

1. In Siebel Tools, update the HTML Help URL property for all Help Id objects to reflect the correct filename.

   In this example, update the HTML Help URL property for all Help Id objects to be help/siebhelp.hlp.
2. Do one of the following:
   - If you only changed HTML Help URL properties, recompile the Repository Help Id project to create an SRF and implement the changes.
   - If you made other changes, recompile all affected projects to create an SRF and implement the changes.
3. Distribute the repository file and the help file (Windows compiled help file) to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

   For example, for Siebel Web Client implementations, distribute the repository file and copy siebhelp.hlp to the correct Siebel Server location (<*install dir*>\public\<*install language*>\help).

Customer Applications

The topics in this section describe the details of the help implementation in Siebel 7 customer applications. It explains the location of the help files and provides information about using Siebel Tools to verify the calls to the online help. In addition, instructions are included for customizing the help content and migrating your existing customized online help from earlier versions.

The following topics are included:

Location of Customer Application Help Files

The location of the help files is determined by the location where you installed your Siebel applications.

Help files are installed in the following location on the server:

<*install dir*>\public\<*install language*>\help, where

- *install dir* is the directory where you installed the Siebel Web Server Extensions
- *install language* is the language you selected during installation

During the installation process, your Web server is configured so that
<*install dir*>\public\<*install language*>\
becomes the root directory for the URL
http://<*hostname*>/<*Siebel application name*>

When the Value property of the Url Web Page Item Parameter maps to a URL, that URL is relative to http://<*hostname*>/<*Siebel application name*>.

For example, if you are running Siebel eSales on the server siebsrvr, when you request help, the page that will appear is
http://siebsrvr/esales/help/comgeneric.htm. This means that you can put the help files in any directory that can be referenced from
http://<*hostname*>/<*Siebel_application_name*>.

For example, assume that in <*install dir*>\public\<*install language*>\, you create a directory called customizedhelp and you create a start page for your help system, called myonlinehelp.htm. After that, you configure your Web server so that this directory is exposed to your application users. In this case, you would create a Web Page Item Parameter object definition that points to the URL customizedhelp; the Web Page Item Parameter object properties would be:

Property	Value
Name	Url
Value	customizedhelp/ myonlinehelp.htm

Online Help and Siebel Tools

In Siebel Tools, a container Web Page object is used to establish the link between an application and its help start page, as shown in Figure 137.

Each Application object has a Container Web Page property which represents a Web Page object. Each Web Page object contains Web Page Item objects. The HelpButton Web Page Item has one parameter that identifies the start page of the help system for the application.

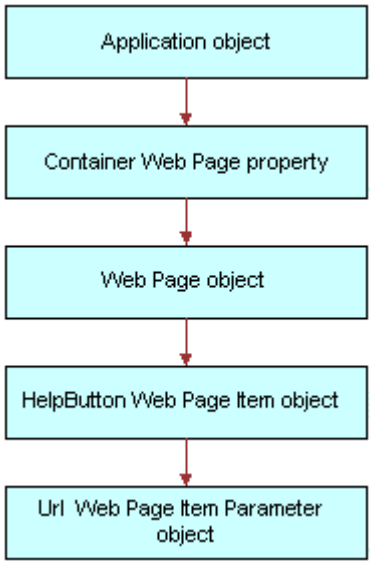


Figure 137. Establishing the Link Between an Application and Help

Changing Help Links

If you want to use a different start page for the help, you can change the help link for the application.

To change a help link

1. In Siebel Tools, in the Object Explorer (Types tab) select the Application object type.
2. In the Object List Editor, find the Container Web Page that the application uses.
3. In the Object Explorer (Types tab), select the Web Page object type.
4. In the Object List Editor, find and select the Web Page object definition that uses the Container Web Page.
5. In the Object Explorer (Types tab), expand the Web Page object type and select the Web Page Item object type.
6. In the Object List Editor, select the HelpButton Web Page Item object definition, and in the Object Explorer (Types tab), expand the Web Page Item object to expose and select the Web Page Item Parameter object type.
7. In the Url Web Page Item Parameter object definition, change the Value property to reflect the name of the HTML file you want to use as a start page.

   For example, help/index.htm.
8. Recompile to create an updated SRF.
9. Optionally, in the HTML file you want to use as a start page, add a reference to the Siebel help cascading style sheet (siebhelp.css) and add the necessary code to implement the navigation buttons.

   NOTE: You can use help/siebcomgeneric.htm as an example.
10. Distribute the repository file and the new HTML file to the appropriate Siebel Server.

Adding Help Links for New Applications

If you create a new application using Siebel Tools, you can add help links to your application.

To add a help link

1. In Siebel Tools, in the Object Explorer (Types tab) select the Application object type.
2. In the Object List Editor, find the value of the Container Web Page property for the Application object.

   For example, the Container Web Page property of the Siebel eSales Application object is CC Container Page (eSales). Container Web pages map to the Web Page object in Siebel Tools.
3. In the Object Explorer (Types tab), select the Web Page object type.
4. In the Object List Editor, find and select the Web Page object definition that uses the Container Web Page.
5. In the Object Explorer (Types tab), expand the Web Page object type to expose the Web Page Item object type.

6 In the Object List Editor, create a new Web Page Item object definition with the following properties:

Property	Value
Name	HelpButton
Type	Link
Caption	Help
Method Invoked	GotoURL
Item Identifier	A number between 11 and 19 that is not used by another Web Page Item object definition.
HTML Attribute	target="_blank"

7 In the Object List Editor, select the Web Page Item object definition you created, and in the Object Explorer (Types tab), expand the Web Page Item object to expose the Web Page Item Parameter object type.

8 Create a Web Page Item Parameter object definition with the following properties:

Property	Value
Name	Url
Value	help/*siebcomgeneric*.htm siebcomgeneric.htm is the default start page for the help system. You can change this parameter to point to a different file, for example, index.htm.

9 Recompile to create an updated SRF.

10 Distribute the repository file to the appropriate Siebel Server.

11 If you are using a different start page, test your changes and upload the new HTML file to the Siebel Server.

For more information, see "Testing and Distributing Changes" on page 579.

Customizing Help Content

Since the help consists of HTML pages, you can use any HTML editor to change the content of a help topic.

To customize help content

1 Find the HTML file you want to change.

   For a list of source files, see "Help Source Files."

2 Open the HTML file, make your changes, and save the file.

   NOTE: If a redirect tag exists in the file, be sure to remove the tag. The following is an example of a redirect tag:

```
<META HTTP-EQUIV="REFRESH" CONTENT="1; URL=siebcomgeneric.htm">
```

3 Test your changes and update the HTML file on the Siebel Server.

   For more information, see "Testing and Distributing Changes" on page 579.

Adding Help Content

You can add help content in the existing HTML files, as explained in "Customizing Help Content" on page 599, or you can add new HTML files to complement the existing content. If you add new HTML files, you must add links to these new files from the existing files to make sure that your users will be able to access the content.

To add help content by adding HTML files

1 Create the new HTML file with your content.

   TIP: Start with one of the files included in the Siebel help system to avoid losing style specifications.

2 Test your changes.

   For more information, see "Testing and Distributing Changes" on page 579.

3 Upload the HTML file to the Siebel Server.

   For the exact location of the help files, see "Location of Customer Application Help Files" on page 596.

Migrating Help

You can update the online help with customized information contained in your help system from a previous Siebel release by using the Siebel topic files included with your Siebel eBusiness Applications.

To update the Siebel HTML file with information from a customized HTML file

1 Find the name of the HTML file to update and open the file.

   See "Customer Applications Files" on page 604 for a complete list of files.

2 Open the HTML file used to create help content in the previous version.

   If you used the Siebel filename in earlier releases, the filename would be Siebel_eBusiness_Help.htm.

3 Use copy and paste functionality to update the Siebel HTML file.

4 Apply appropriate HTML tags to format content and save the HTML file.

   NOTE: If a redirect tag exists in the file, be sure to remove the tag. The following is an example of a redirect tag:

```
<META HTTP-EQUIV="REFRESH" CONTENT="1; URL=siebwelcome.htm">
```

5 Test your changes and update the HTML file on the Siebel Server.

   For more information, see "Testing and Distributing Changes" on page 579.

Global Deployment

The topics in this section provide high-level information about deploying online help in different languages.

The following topics are included:

For additional information about global deployment, see *Global Deployment Guide*.

Language Folders

Your Siebel application comes with localized online help. Localized online help files are located in the language-specific folders on either the server or the Mobile or Dedicated Web client.

Help files are installed in the following location on the server:

<*install dir*>\public\<*install language*>\help, where

- *install dir* is the directory where you installed the Siebel Web Server Extensions
- *install language* is the language you selected during installation

For details about the location of the help files for different Siebel applications, see "Location of Employee Application Help Files" on page 579 and "Location of Customer Application Help Files" on page 596.

Localizing Online Help

If you are deploying your Siebel application in a language not available from Siebel Systems, Inc., and you want to deploy online help in that language, you must localize the online help. Figure 138 shows the typical steps involved in help localization.

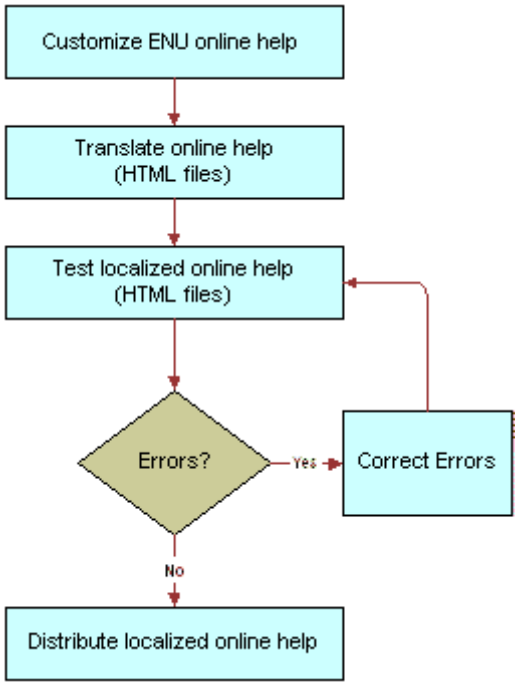


Figure 138. Help Localization Steps

To localize online help

1. (Optional) Customize the ENU (American English) help to suit your implementation.
2. Translate the HTML source files (that make up the help system) by modifying the flat files directly.
3. Test and distribute the localized online help to the appropriate Siebel Servers, Siebel Mobile Web Clients, and Siebel Dedicated Web Clients.

Help Source Files

The following topics list the source files that make up the help system for the employee, partner, and customer applications.

■ "Employee Applications Files" on page 603
■ "Customer Applications Files" on page 604

■ "Cascading Style Sheet" on page 605

Siebel Systems, Inc. provides the text of the application online help in a format that you can edit to reflect your unique implementation. You can use the content described in this document as the foundation for the help system you distribute to users.

Note that information about your Siebel application (including the online help) is covered by your confidentiality agreement with Siebel Systems and thus cannot be shared with individuals who do not already have access to your Siebel implementation.

Employee Applications Files

The help system for employee applications consists of HTML files (listed under HTML Files), image files, and a cascading style sheet (siebhelp.css, described in "Cascading Style Sheet" on page 605).

HTML Files

In the HTML files that make up the help system, there are many files that have generic content (based on siebendusegeneric.htm). With the files already in place, you can simply use the existing file to customize the content to suit your implementation. The following table lists the files that do not have generic content.

Filename	Description
siebbasicscontrols.htm	Using Field Controls
siebbasicscontrolscal.htm	Selecting Date and Time Information
siebbasicscontrolscurr.htm	Using the Currency Calculator
siebbasicsdata.htm	Working with Data
siebbasicsdatatoc.htm	Table of contents: Working with Data
siebbasicskbshortcuts.htm	Using Keyboard Shortcuts
siebbasicsnav.htm	Navigating the Application
siebbasicsselectiondbs.htm	Using Selection Dialog Boxes
siebcalendar.htm	Using the Calendar
siebcharts.htm	Using Charts
siebcorrespondence.htm	Using Siebel Correspondence
siebcorrespondencetoc.htm	Table of contents: Using Siebel Correspondence

Filename	Description
siebfindinginfotoc.htm	Table of contents: Finding Information
siebhomepage.htm	Using the Home Page
siebhomepagetoc.htm	Table of contents: Using the Home Page
siebindex.htm	Help Index
siebliterature.htm	Working with Literature
siebliteraturetoc.htm	Table of contents: Working with Literature
sieboptions.htm	Setting User Options
sieboptionstoc.htm	Table of contents: Setting User Options
siebpaging.htm	Siebel Paging help topics
siebquery.htm	Using Queries
siebquerytoc.htm	Table of contents: Using Queries
siebreports.htm	Working with Reports
siebsearch.htm	Using the Siebel Search Center
siebsearchtoc.htm	Table of contents: Using the Siebel Search Center
siebstarthelp.htm	Start page (main table of contents)
siebsynch.htm	Synchronizing Data
siebsynchtoc.htm	Table of contents: Synchronizing Data
siebuserinterfacetoc.htm	Table of contents: Understanding the User Interface
siebusingcalendartoc.htm	Table of contents: Using the Calendar
siebwelcome.htm	Welcome to Siebel Online Help

Customer Applications Files

The help system for customer applications consists of HTML files (listed in HTML Files), image files, and a cascading style sheet (siebhelp.css, described in "Cascading Style Sheet" on page 605).

HTML Files

The following table lists the HTML files that make up the customer application help system.

Filename	Description
siebcomcontactus.htm	Contacting Us
siebcomfindinginfo.htm	Finding Information
siebcomgeneric.htm	Start page (main table of contents)
siebcomgenericcontactustoc.htm	Table of contents: Contacting Us
siebcomgenericfindinginfotoc.htm	Table of contents: Finding Information
siebcomgenericlogintoc.htm	Table of contents: Logging In
siebcomgenericregistrationtoc.htm	Table of contents: Registration
siebcomgenericuserinterfacetoc.htm	Table of contents: Understanding the User Interface
siebcomloggingin.htm	Logging In
siebcomregistration.htm	Registration
siebcomsearch.htm	Using Search
siebcomupdatinguserprofile.htm	Updating Your User Profile
siebcomuserinterface.htm	Understanding the User Interface
siebcomwwattachments.htm	Viewing Attachments

Cascading Style Sheet

Siebel online help uses a cascading style sheet (siebhelp.css) to control the appearance of the help pages. A cascading style sheet includes typographical and formatting information on how the Web page should appear, such as the text font. A cascading style sheet gives the author control over the appearance of the page.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

W

X

Z