SIEBEL 7
eBusiness

CONFIGURATION GUIDELINES

MIDMARKET EDITION

VERSION 7.5

12-BD362J

SEPTEMBER 2002

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

Printed in the United States of America

The full text search capabilities of Siebel eBusiness Applications include technology used under license from Hummingbird Ltd. and are the copyright of Hummingbird Ltd. and/or its licensors.

Siebel, the Siebel logo, TrickleSync, TSQ, Universal Agent, and other Siebel product names referenced herein are trademarks of Siebel Systems, Inc., and may be registered in certain jurisdictions.

Supportsoft™ is a registered trademark of Supportsoft, Inc. Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

U.S. GOVERNMENT RESTRICTED RIGHTS. Programs, Ancillary Programs and Documentation, delivered subject to the Department of Defense Federal Acquisition Regulation Supplement, are "commercial computer software" as set forth in DFARS 227.7202, Commercial Computer Software and Commercial Computer Software Documentation, and as such, any use, duplication and disclosure of the Programs, Ancillary Programs and Documentation shall be subject to the restrictions contained in the applicable Siebel license agreement. All other use, duplication and disclosure of the Programs, Ancillary Programs and Documentation by the U.S. Government shall be subject to the applicable Siebel license agreement and the restrictions contained in subsection (c) of FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987), or FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987), as applicable. Contractor/licensor is Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404.

Proprietary Information

Siebel Systems, Inc. considers information included in this documentation and in Siebel eBusiness Applications Online Help to be Confidential Information. Your access to and use of this Confidential Information are subject to the terms and conditions of: (1) the applicable Siebel Systems software license agreement, which has been executed and with which you agree to comply; and (2) the proprietary and restricted rights notices included in this documentation.

Contents

Chapter 3. Specialized Classes

Chapter 4. Creating and Modifying Objects

Chapter 5. Performance Guidelines

Chapter 6. Development Standards for Siebel Script Languages and Object Interfaces

Chapter 7. Configuring the User Interface

Chapter 8. Managing Repositories

Chapter 9. Visibility

Chapter 10. Database Extensibility

Index

Introduction

This document gives a number of guidelines you can apply when configuring or customizing the Siebel development repository. Although this document includes many recommendations, they do not apply to all environments. You should consider them a starting point for creating a well-designed overall application.

How This Guide Is Organized

A successful architecture will help make the Siebel application implementation a success, while a poor architecture will undermine an otherwise effective configuration. Also, properly configuring your application gets the most out of the architecture that you have in place. Poor configuration will likely cause, for instance, poor performance from your staff or your servers.

The purpose of this document is to make sure that any configurations you perform:

- Are accurate.
- Provide an intuitive user interface.
- Are easy to maintain and upgrade.
- Maintain optimum performance of the Siebel eBusiness Application.

The document is not intended to give step-by-step instructions on how to perform specific configurations.

Technical professionals and computer programmers can use this document to configure Siebel eBusiness applications. It is recommended that your project team members attend Siebel University's Siebel eBusiness Essentials course before beginning configurations.

Revision History

Configuration Guidelines, MidMarket Edition, Version 7.5

Setting Up Developers 1

This section gives details about setting up developers.

Organizing Projects

A project is a named set of objects used to group related object definitions. Projects can be checked in, checked out, locked for modification, and compiled.

Follow these guidelines when working with projects:

- Projects should contain related object definitions. Because projects are typically organized by functional area or object type, keeping related object definitions together helps make sure dependent code is checked in and out at the same time.
- When working in a multiuser environment, try to organize tasks around projects.
- Create new projects for any major new areas of configuration, such as business objects or business components.
- Try to minimize the size of new projects. This decreases both the time to compile and the check-in and checkout processes. Minimizing the size of a project also results in fewer conflicts between developers working on different objects within the same project.

Using Local Databases

Although Siebel Tools lets developers operate in a client/server mode with the development database server, developers should never make configuration changes directly in the server database, for the following reasons:

- When working directly in the server database, you cannot undo or back out undesired changes to a specific project. The only option is to recover the entire repository.
- Your changes are immediately available to other developers when they check out projects or compile an SRF in the database server. If you have not completed work on the project, these incomplete changes can cause problems for other developers.

To prevent these problems and promote efficient team-based development, each developer should be configured as a mobile user, with a local SQL Anywhere or MS SQL database. Initially, each developer should populate the local database with a read-only copy of all projects on the server. When developers need to modify a specific project, they can check out that project from the server database. This locks the project on the server database to prevent other developers from making changes to it, and transfers a modifiable copy of the project to the local database. The developer can then modify and unit-test the project against the local database. After the work is complete, the developer can check the project back into the server database. If a developer wants to discard the local configuration work, the developer can revert the local database copy of the project to the server version, which is kept as an archive file each time a project is checked out.

Preparing Local Development Environments

When setting up developers to work in a local environment, complete these steps, in this order:

1. Install Siebel Tools in the desired folder on developer machines. By default, this is the C:\siebdev directory.
2. Install the Siebel eBusiness application in the appropriate folder on developer machines. By default, this is the C:\Siebel directory.
3. Set up each developer as an employee and mobile client. Using a Siebel eBusiness Applications client connected to the development server database, create an Employee and a Mobile User record for each developer. Use the developer's first and last names for the employee first and last names and a standardized short name, such as first initial plus last name, for the login name. This makes it easy to identify who has locked a project.
4. Grant each developer a position and responsibility. You can grant the Siebel Administrator responsibility to all developers; alternatively, you may want to create a responsibility with access to all views except System, Service, and Marketing Administration to prevent unintended changes to important system preferences and data. 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.
5. Schedule a Generate New Database Component Request to create an extract for the user. This process creates a template for the developer's local database that is populated with business data only, not with repository data. All enterprise visible data is extracted into this template, together with any limited visibility data (such as Contacts, Accounts, or Opportunities) to which this user has access.
6. Connect to Siebel Tools locally to initialize the developer's mobile client database. Specify the Siebel developer login created in Step 3 with an appropriate password. The initialization program creates the local database (sse_data.dbf) in the C:\SIEBDEV\LOCAL directory.

7 Do an initial checkout (get) of all projects, which retrieves all projects but does not lock them. Developers can then individually check in or check out their projects.

   Use the Check In and Check Out commands to copy projects between the local database and the server. When you check out a project, a copy remains on the server but the project is now locked on the server. Other developers cannot make changes to projects that are locked on the server.

When you check in a project, it is copied from your local database to the server database, and it replaces the existing definition of the project on the server. To check in a project, you must have it locked on the server database (that is, you must have first checked it out).

NOTE: When you check in or check out, the project on the target database is overwritten with the new project.

Doing an Initial Checkout (Get)

After creating each developer's local database, you must do an initial checkout to populate the local database with a read-only copy of all projects from the repository. The developer can then compile an SRF file against the local database; this is required to fully test changes locally. As previously mentioned, the newly initialized local database contains only a skeletal repository without any projects. The following procedure describes how to check out (get) all projects to a developer's local database.

To check out (get) all projects to a local database

1 In Siebel Tools, choose Tools > Check Out.

2 Select your development repository.

   The name of your development repository should follow your repository naming conventions. For more information on repository naming conventions, see Chapter 8, "Managing Repositories."

3 Make sure the Server and Client Data Sources point to the correct databases.

If the Server and Client Data Sources are *not* pointing to the correct databases, click Options and navigate to the correct databases.

NOTE: Make sure All Projects is checked.

4 Click Get.

After you check out all projects, your local database now contains the initialized repository data. By default, the new repository opens with the name Siebel Repository. After an upgrade, however, there may be multiple repositories in the database; if so, you must manually open the Siebel Repository, as described in the following procedure.

To manually open the Siebel Repository

1 In Siebel Tools, choose File > Open Repository.

2 From the Open Repository dialog box, select the repository you want to open, and click Open.

You are now ready to begin your development using Siebel Tools.

Checking Out Projects for Modification

Before developers can modify a project on the local database, they must first check out and lock a copy of the project on the server database.

To check out and lock a project

1 In Siebel Tools, choose Tools > Check Out.

2 Make sure the correct repository is selected.

NOTE: This should be the same repository that you opened in the previous section.

3 Make sure the Server and Client Data Sources point to the correct databases.

   If the Server and Client Data Sources are *not* pointing to the correct databases, click Options and navigate to the correct databases.

4 Select the projects that you want to modify.

5 Click Check Out.

   The project is now locked on both the local and server databases.

Checking Out Projects For Refresh

In a multiuser development environment, such as Siebel Tools, developers frequently need to check out projects modified by other developers to update, or *refresh*, their local development environment. If you are only checking out a project to refresh your local development environment, do an initial checkout (get) and *not* a standard checkout, as described in the following procedure.

To check out projects for refresh

1 In Siebel Tools, choose Tools > Check Out.

2 In the Check Out dialog box, select the specific projects you want to refresh.

NOTE: You can also select Updated Projects to retrieve all updated projects.

3 Click Get.

NOTE: To be sure that you have a valid local configuration, make sure that all the projects modified by other developers are checked in before doing the get. You should do this usually once a day to make sure your repository is as current as possible.

Checking In Projects

You can only check in projects that you have locked through a checkout process. Before checking in a project, be sure that the project is stable and has been thoroughly tested against your local database. Check in a project only after completing all dependent code.

You should check in all dependent projects at the same time to be sure that the server configuration remains consistent. Also, consider how you time your check-in and how that impacts the work of other developers. In some instances, you may need to check in a project before you have fully completed the configurations required in that project. For example, if another developer's configurations depend on a particular feature you have added to your project, you may need to check in your project before configuring other features. This lets other developers test their configurations with your new feature. Plan carefully to be sure that you complete the dependent configuration before starting other independent configurations. Alternatively, if the particular feature is simple to implement, then other developers can lock that project locally and manually apply the required changes (or use an SIF archive file). In this case, the developer cannot check in the locally locked project because another developer has the project locked on the server.

To check in your projects

1. In Siebel Tools, choose Tools > Check In.

   The dialog box displays a list of all your locked projects.

2. Select the individual projects you want to check in or select Locked Projects to select all locked projects.

   NOTE: Select Locked Projects/New Projects only when working with complete and stable configuration work.

3. Check Maintain Lock to check in projects but keep them locked on the server. (Optional)

4. Make sure the Server and Client Data Sources point to the correct databases.

   If the Server and Client Data Sources are *not* pointing to the correct databases, click Options and navigate to the correct databases.

5. Click Check In.

Canceling Changes to a Checked Out Project

Occasionally, you may want to discard the changes you made to a checked out project. For example, you might decide that it is easier to start over instead of undoing previous changes. There is no Undo Check Out command. Instead, you must check out the project from the server database again, as described in "Checking Out Projects for Modification" on page 16. This lets you overwrite your existing copy of the project with the copy of the project that is on the server. However, repeating the checkout procedure will not remove the server lock. If you want to unlock a project on the server you must check in the project and then check it out again.

If you need to revert back to the project's original state but cannot connect to the server to check out the project, you can use an archive file (SIF) of the project, which Siebel Tools automatically creates each time you check out a project. This archive file is stored in the Siebel Tools temp/projects directory and can be imported the same way as other SIF files.

Naming Conventions 2

This section provides naming conventions for Siebel eBusiness Applications.

Naming Conventions

Table 1 lists the naming conventions you should use when configuring your Siebel application.

Table 1. Naming Conventions for Siebel Configurations

Object	Convention
Any	For any new object you create (top-level or child), prefix the name with the company name (or its acronym). For example, if you are creating a new Hobby business component for ABC Corporation, name it ABC Hobby. Similarly, if you are creating a new Description field in the standard Account business component, name it ABC Description. Also consider putting the three-letter acronym at the end of the name instead of the beginning. This convention allows you to search for custom objects by acronym but still see related objects sorted together, such as Approval Authority and Approval Authority Code - XYZ. The only exception to this standard is that when you add a new child object to a parent object that already has a prefix, there is no need to add it again at the child level. For example, if you are adding a Description field to the ABC Hobby business component, name it Description instead of ABC Description.
General	Most object names (except applets) should be singular and not plural. ■ Avoid special characters like slashes, parentheses, periods, and so on in object names (except for slashes in link names). The "#" (number sign) character is a valid character. ■ Avoid prepositions in object names—for example, use # Weeks or just Weeks instead of Number of Weeks. ■ Avoid the word Amount in the names of currency fields—for example, use Revenue instead of Revenue Amount.
Links	Use the convention < parentBusComp > / < childBusComp > . By default, the name for new links is set to this format but you can override the default name.
Join	When adding a new join to an existing standard business component, always give it an alias and make sure the alias is prefixed with the company name or acronym.

Table 1. Naming Conventions for Siebel Configurations

Object	Convention
Foreign Key Fields	These fields should have a name like < Entity > Id. If the foreign key is serving as a primary pointer, it should have a name like Primary < Entity > Id. For a primary pointer to an entity that is already prefixed, prefix the foreign key field with the company name or acronym (as usual), but do not include the prefix in the < Entity > name. For example, a primary pointer to the ABC Subsegment business component would be named ABC Primary Subsegment Id, instead of Primary ABC Subsegment Id or ABC Primary ABC Subsegment Id.
New LOV_TYPE in List of Values	Prefix with the company name or acronym.
Form applets and Profile applets	Titles should be singular.
List applets and MVG applets	Titles should be plural.
Associate applets	Titles should be in the format Add < entities > , where < entities > is plural.
Pick applets	The titles should be in the format Pick < entity > , where < entity > is singular.

Tables

There are over 2,000 database tables in the Siebel Data Model. Each of these tables follows a standard naming convention to help users identify individual tables. For information on naming conventions for tables, see *Siebel Data Model Reference*.

Business Objects and Business Components

The following general naming conventions apply to business objects and business components:

- Business component and business object names must be unique and meaningful. Avoid naming new business components or business objects by adding a number suffix to an existing name (for example, Account 2). This type of name does not clearly communicate how the new object definition differs from the original one. For example, if you create a new business object to support a unique account screen for your Telesales group, you might call the business object Telesales Account to clearly indicate its intended use.
- Name all new business components and business objects with a prefix that identifies your company. For example, ABC Incorporated could name a new object definition ABC Telesales Account. This approach makes it easy to identify the object definitions in the repository that are not object definitions defined by Siebel Systems. This approach also makes it easy to query for your newly defined object definitions and simplifies upgrading your repository.

Initial-capitalize business component and business object names, for example, Account rather than account. This prevents unexpected sorting behavior in the Object List Editor.

Joins

You should use the following convention for naming joins:

Object Type	Name Format	Example
Join	*name of the joined table*	S_ORG_EXT S_CONTACTS

Links

You should use the following convention for naming links:

Object Type	Name Format	Example
Link	*buscomp1/buscomp2*	Account/Account Service Agreement

There are two types of links: one-to-many (1:M) and many-to-many (M:M).

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.

In addition, note the conventions in Table 2 for specific view types.

Table 2. 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

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 3.

Table 3. 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

Controls and List Columns

A control (except for a button, prompt, or system control) must correspond to a field on the business component on which the applet is based. The control's Name property should have the same value as the field's Name property.

Follow these guidelines when creating display names:

- Use the same display name for an underlying field in every applet in which it appears.
- Avoid using abbreviations when enough room is available for you to spell out the word. For example, when there is sufficient space, use *Opportunity* instead of *Oppty*.
- If you must abbreviate, use the same abbreviation throughout the application. For example, always use *Account Num* and do not switch between *Account Num, Account No.,* and *Account #*.
- Initial-capitalize control and list column names, for example, Account Num rather than account num. This prevents unexpected sorting behavior in the Object List Editor.

Here are other naming considerations:

- 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.
- 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. Also, note the unusual capitalization of the word PickList as it is used throughout the standard repository.
- 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.
- Make sure read-only applets always have the string Read-Only immediately before the word Applet in their name—for example, ABC Account List Read-Only Applet instead of ABC Account List Applet - Read-Only.
- Make sure read-only views always have the string Read-Only immediately before the word View in their name—for example, ABC Account Common Profile Read-Only View instead of ABC Account Common Profile View - Read-Only.
- 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.

- 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.

- Always name multi-value group applets < BusComp > Mvg Applet. (Note the case sensitivity of Mvg.)
- Always name Pick applets < BusComp > Pick Applet. Always name association applets < BusComp > Assoc Applet.
- Always name a profile applet < Entity > Profile Applet, instead of < Entity > Profile Form Applet.
- 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.
- When creating a physical extension column:
  - Use an _ID suffix for foreign key columns (varchar 15) and a _CD suffix for List of Values domain fields (varchar 30).

    NOTE: Do *not* use these suffixes for fields that do not meet this criteria.

  - Use a physical type of varchar 40 for phone number fields.

- Remember that an extension column on a base table is automatically given an X_ prefix, so it is not necessary to add an additional prefix (for example, X_ABC_) to distinguish the columns from a standard column.
- Limit column names to 18 characters, which is the Siebel standard. Using this standard allows the application to use the column names as the basis for related column names without ever approaching the database limit for column names.
- On MS SQL Server 7 or Oracle, always extend a base table in favor of using an _X extension table. Note that this was not feasible on MS SQL Server 6.5 (and previous versions) or Sybase SQL Server because page size and the number of columns per table were limited.

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.

Specialized Classes 3

This section contains information on specialized classes.

Specialized Classes

A class property gives an object access to specific functionality. The class refers to a dynamic-link library (DLL) that is installed on the client or server machine and holds the required functionality. You can use Siebel Tools to view Class and DLL objects. If you need to make these objects visible in Siebel Tools, see *Siebel Tools Reference, MidMarket Edition*.

Because the functionality for specialized classes is not published, you must understand what specialized classes represent and how to use them most effectively. Using the class property incorrectly can cause issues whose cause is difficult to determine. For example, records may be added randomly to the database, records may be deleted, or run time errors may occur. In general, the class property should be used with extreme care.

The following objects have a class property and can be configured using Siebel Tools:

- Applets
- Business Components
- Business Services
- Controls
- Reports
- Search Engines
- Toolbars

NOTE: You need to use only the class property of an object when copying existing business components or applets.

Most of the standard business components have a class property of either CSSBusComp or CSSBCBase. The most fundamental class is CSSBusComp, from which CSSBCBase is derived. All other classes used for business components are derived from one of these two classes. This base class provides common functionality across all business components, including record navigation, business component events, undo, merge, search, and sort. CSSBCBase adds functionality to CSSBusComp that is associated with using the Siebel State Model, and allows the use of user properties such as Deep Copy and Deep Delete. It is recommended that you use CSSBusComp for all new business components and CSSBCBase if expected functionality is not available. CSSBusComp and CSSBCBase are not classified as specialized classes.

The rest of the supplied business components are based on classes that are derived from either CSSBusComp or CSSBCBase. For example, the Person Forecast business component has the class property set as CSSBCRevenueForecast, which is derived from CSSBCMstrFcstChild, which in turn is derived from CSSBusComp. You can view these relationships in Siebel Tools. This class dictates that each time users submit their current forecast, a new forecast record is created. The class controls this behavior and it cannot be altered through configuration. These remaining classes are considered specialized classes.

When you copy a business component, understand that you are not only copying fields, MVLs, or Pickmaps, but also specialized functionality, of which you cannot be certain.

The following guidelines can help you determine if and when you need to adjust a class property.

Business Components

Follow these guidelines when copying or creating business components:

- Avoid copying business components based on specialized classes unless you are trying to get exactly the same functionality with minimal changes.
- If you copy a business component and alter the class to CSSBusComp or CSSBCBase, be aware that this business component can now access the data without the special rules active. Be sure that this is what you want because specialized rules may not be followed.
- By default, when you create a business component, it has a class property of CSSBusComp. You can change this to CSSBCBase if required functionality, such as Deep Copy, is not available. Do not change the class property to any other value.
- If a business component is based on a specialized class, do not delete, inactivate, or remap any of the standard child objects. If the object is not required for your configuration, configure the user interface to not show, in the case of a field, any controls which map to that field.
- Only copy a business component based on a specialized class if you require a minor change to the existing business component and also need to use the original business component for something else.

  NOTE: If you require a read-only version of the business component, create duplicate applets and use a No Delete setting, instead of duplicating the business component.

- When copying a business component based on a nonspecialized class, remove all redundant objects. Be aware that some of these objects may be referred to by properties of the business component or other child objects.
- If you need to copy a business component and keep the specialized class, do not delete or inactivate any of the fields that are present in the copy. Do not display the fields on applets if you do not need them. Because the specialized class may reference these fields, you must keep the original settings. Also, be sure to set the Upgrade Ancestor property, so the copy will upgrade correctly during the upgrade process.

Applets

Follow these guidelines when copying or creating applets:

- When you create a new applet, make sure you set the class property correctly. The type of applet you require determines the appropriate value for the class property. To simplify the creation of the many records and settings necessary to make an applet function correctly, Siebel Tools has a number of wizards that help simplify the process. For information on accessing these wizards, see *Siebel Tools Reference, MidMarket Edition*. The following table lists the default class for each type of applet:

Applet Type	Class
Chart	CSSFrameChart
Form	CSSFrame
List	CSSFrameList
MVG	CSSFrameList
Pick	CSSFrameList
Tree	CSSFrameTree

- You should never need to alter the class property for an applet. The default settings have been preconfigured to allow the applet to function correctly.
- To create functionality on an applet that is identical to another applet, copy the applet whose functionality you want to mimic and adjust the copied applet to fulfill your requirements. However, be aware that in some cases the class for the applet and the class for the business component work together to provide the functionality. For example, the Campaign Contact/Prospect applet has a class property of CSSFrameCampaignContactList. The associated business component is Campaign List Contact, which maps to a specialized class of CSSBCCampaignContact. In this case, the class for the business component and the class for the applet work together to provide certain functionality, such as the functionality for the Create Opportunity button.

Creating and Modifying Objects 4

This section provides guidelines for the creation and modification of data object layer, business object layer, and user interface layer objects.

Usage and Configuration of Nonlicensed Objects

The licensing agreement between Siebel Systems 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.

Object Definitions Not To Be Reconfigured

Though Siebel Tools is implemented as applets, views, and so on in the Siebel repository, Siebel Tools as an application should not be reconfigured in the manner of other applications such as Siebel Sales and Siebel Service. Making modifications may cause application errors, and make your configurations difficult for Siebel Technical Support personnel to understand and support.

In particular, object definitions whose names start with the word Repository, such as the applet called Repository Business Component List Applet and the view called Repository Chart Element List View, should not be customized.

NOTE: Repository Business Components are for Siebel Systems internal use only and their customization is not supported. These business components are primarily in the Repdetd Project and the CSSBCReposObj Class.

A number of object types such as Class, String Map, Type, DLL, and Attribute are reserved. These are nonconfigurable object types, and are listed in *Siebel Tools Online Help, MidMarket Edition*. Do not delete, modify, or add object definitions of these types.

Inactive object definitions in the repository are not supported for customer use. The repository contains some inactive object definitions, some of which are obsolete and some of which are under construction, and may be removed or implemented in a future release. Do not activate object definitions unless you are reactivating those that you have previously deactivated.

NOTE: Tables in the data model and utility SQL scripts also may not be available for customer use, even though they are present in standard Siebel applications.

Supporting Business Logic in Siebel Applications

Siebel Tools Reference, MidMarket Edition describes many ways to configure a Siebel application using Siebel Tools. However, Siebel eBusiness Applications include several types of functionality that may work just as well as a custom configuration, to support the mechanisms of business logic.

Whenever possible, you should use existing Siebel application functionality. Instead of writing many scripts or making other configuration changes, you should use the varied functionality built into Siebel products such as Business Process Designer, Personalization, SmartScript, Territory Assignment Manager, and State Model. Because these areas of functionality are set up in the client administrative views rather than in Siebel Tools, they are more likely to be overlooked by application developers.

This section provides cross-references to more information.

For Development Theme...	Refer to...
Integration	*Business Processes and Rules: Siebel eBusiness Application Integration Volume IV, MidMarket Edition*
	Siebel Enterprise Integration Manager Administration Guide, MidMarket Edition
	Overview: Siebel eBusiness Application Integration Volume I, MidMarket Edition
Application Development	*Siebel Tools Online Help, MidMarket Edition*
	Siebel Reports Administration Guide, MidMarket Edition
	Developing and Deploying Siebel eBusiness Applications, MidMarket Edition
System Administration	*Siebel Territory Assignment Manager Administration Guide, MidMarket Edition*
	Siebel Business Process Designer Administration Guide, MidMarket Edition
Application Administration — Employee Applications	*Applications Administration Guide, MidMarket Edition*
	Personalization Administration Guide, MidMarket Edition
	Siebel SmartScript Administration Guide, MidMarket Edition
Customer and Partner Applications	*Pricing Administration Guide, MidMarket Edition*
	Product Administration Guide, MidMarket Edition

Creating Objects

At times, you are required to create a new object — for example, when copying a view to make a read-only version of it for a specific responsibility, or when copying an applet that is used by one group of users and removing list columns to meet a requirement for a different group of users. In these cases, copy an existing object and make the necessary additions. If you want to upgrade the object, you need to specify the Upgrade Ancestry property for all of the relevant business components and applets. This strategy saves development time and makes it easier to set the relevant properties for that object type.

When creating objects, you must proceed in a particular order. This helps you make sure the correct values for all required properties are available as options. For example, you should create business components and related links before creating a business object as shown in Figure 1. You should create all your data manipulation objects before your presentation objects.

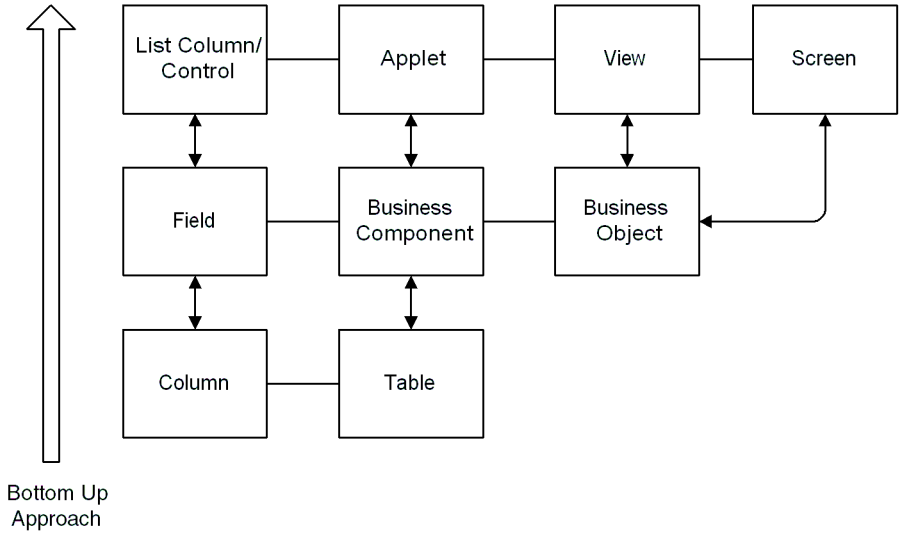

Figure 1. Creating Objects Using a Bottom Up Approach

Modifying Objects

There are many reasons to modify an existing object. For example:

- Many existing objects have been configured for best performance; cloned 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.

New functionality is often added to core business components during a major release. Typically, this new functionality depends on the existence of new fields or joins that have been added to existing business components. During the upgrade process, these new fields or joins will only be added to the standard business components (identified by name) or to copies that are directly related to a standard object. This is why you must specify the Upgrade Ancestry property for any cloned business components, applets, integration objects or reports.

By specifying the Upgrade Ancestry property, you can be sure that, during the upgrade process, all of your cloned business components will match the standard business components that were used as the basis for the cloned object. This helps reduce any difficulties after the upgrade, which are inherently hard to resolve. Many of these errors occur because some C++ code from the specialized class for the business component or applet is set to a field that does not exist in your custom copy of the specialized business component or applet. The only way to resolve the error is to compare your custom business component with the standard business component and manually add any necessary new fields.

NOTE: Clone an object only when you are customizing the application. *Do not* clone objects to maintain a *pristine* copy of the object definition. The Archive feature in Siebel Tools lets you archive one, multiple, or all objects within a project into an SIF file. You can reimport these archive files at any time, and by using a third–party source control program, you can use these archives to control the versions of your configuration changes.

Also, be aware of the following:

- 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 may also create a significant amount of redundant configuration.

Copying Versus Modifying Object Definitions

The supported practice for creating new business components and applets is to reuse and modify standard object definitions. If, instead of using this supported method, you use the Copy Record option to copy standard business components and applets, you run the risk of causing problems during subsequent upgrades of Siebel applications. Some of the reasons why copying can cause problems include:

- **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 results in problems following the upgrade, and these are 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 creates redundancy.** Creating new copies of business components and applets results 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 increases, not reduces, difficulties.** Developers sometimes 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 possibly 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.

In short, modify existing object definitions wherever possible, and avoid using the Copy Record option except when it is truly needed.

However, there are a few special situations in which you should legitimately make a copy of an existing object definition. Some of these situations are described in detail in the subsections below. As a general rule, unless you are certain that you need to make a copy of an object definition, modify rather than copy an existing object definition.

Creating Specialized Objects

This section gives several important reasons why you should not create redundant objects in your configuration through copying objects or cloning specialized classes of objects.

Avoiding Redundant Objects

There are two reasons not to copy objects:

- Copying creates redundancy.

  New copies create redundancy in the repository. If you rename a business component and use it in several views, then new applets must be based on the new business component. This process adds considerable redundancy and effort to the configuration.

- Copying increases difficulties.

  Copying an object will *not* make it easier to upgrade because the object does not change during the upgrade. However, it is easier to remove any controls or list columns that were added during the upgrade than it is to try and manually upgrade the objects by adding all the new functionality from the upgrade.

  For example, if you need a business component to appear in a business object more than once, or if a business component requires a different search specification or predefault property values, you may have to create new business components. You would also need to create a new business component if you create a completely new logical entity that serves a role that is not handled by any existing objects.

Cloning Specialized Objects

Business components and applets all have a class property. This class property is the C++ class, which implements the functionality of the business component or applet. There are generic classes and specialized classes. The generic class for a business component is CSSBusComp; the generic classes for applets are CSSFrame and CSSFrameList. Specialized classes exist for business components or applets that have specialized behavior or features. Examples of specialized modules include Quotes, Forecasting, and Correspondence. For newly created objects, use the generic classes only. New objects take a specialized class only if the new object will be a clone of the original object. In other words, if the specialized behavior associated with using the specialized class is desired in the new object, then it may be appropriate to clone a specialized class.

Be careful when modifying specialized modules and objects like Promotions or Quotes. Often, buttons on these modules rely on being attached to specialized applet classes. (For example, changing a form applet to a list applet may break specialized code). Make only minor changes to specialized objects, such as changing the caption on an applet control or list column. Never rename or delete fields in specialized business components.

Whenever you clone an applet or a business component, populate the Upgrade Ancestor property with the original object's name. For more information, see *Siebel Tools Reference, MidMarket Edition*.

CAUTION: Significantly modifying a specialized object may have severe implications for your application. You must conduct a thorough testing of these objects.

Creating Data Objects

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 *Siebel eBusiness Data Model Reference*.

The standard user interface does not use all the relationships available in the underlying data model; however, most entity relationships are available for developers to use. During the discovery phase of an implementation, you should carefully analyze the business requirements and thoroughly research how to meet these requirements using the existing data model and standard objects.

There is a misconception that if a relationship is not defined in the data model and has not been created in the Business Objects layer, a custom relationship must be created with a custom foreign key. This is not always necessary and is discouraged for these reasons:

- When planning or implementing Mobile Web Client users, be aware that downloading data to the local database is governed by Dock Object Visibility rules. (For more information, see Chapter 9, "Visibility.") 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.

- If you are using Enterprise Integration Manager (EIM) to populate data in Siebel base tables, you can follow this process:
  - Add the custom FK extension column.
  - Add a corresponding column to the EIM table that populates the table with the custom foreign key.
  - Map them together.

  Note, however, that EIM treats the foreign key extension column as a Varchar(15) column instead of a true foreign key. EIM does not maintain referential integrity so resolving the foreign key is a manual process.

  Changing the user key is not supported because it involves extensive changes to EIM mappings and continuous maintenance through upgrades. It also breaks the EIM process.

  You can add custom indexes to include extension columns for increasing performance; however, contact Siebel Expert Services to make sure these indexes do not inadvertently degrade performance.

NOTE: Wherever possible, use the standard relationships available within the Siebel data model. This helps minimize any remote visibility and EIM issues.

Using the Docking Wizard

Before using the Docking Wizard, do the following:

- Analyze configuration workarounds that allow the desired functionality or docking to take place without creating any new docking rules. Siebel Expert Services will analyze any future concerns about new designs and requirements for custom foreign keys.
- If you decide that no additional configuration is required, then analyze whether the docking needs to be bidirectional. If not, contact Siebel Expert Services to have one of the new rules inactivated.
- Be aware that any new Dock Object Visibility rules will have the Check Dock Object Visibility strength of 100 and a resulting visibility strength of 50. If you need to alter these strengths to cause additional or fewer records to dock, submit a Non-Standard Change Request (NSCR) to Siebel Expert Services. For more information on dock objects and the Docking Wizard, see *Siebel Tools Reference, MidMarket Edition*.

In some cases, you can change the relationships between entities. For example, there are objects within the repository that can change the relationship between Contacts and Activities from 1:M to M:M. Although only one relationship at a time is used to maintain data integrity, the data model does provide flexibility to meet your business requirements. You should verify that entity relationships can be changed with existing objects to prevent extensive database schema changes and configuration.

Creating and Modifying Business Objects

A business object groups one or more business components into a logical unit of information. For example, the Opportunity business object may group the Opportunity, Contact, and Product business components as Business Object Components. This exposes the relationships between the data residing in the child business components (such as Contact and Product) with the parent business component, Opportunity. In other words, by grouping these business components within the business object, you can now use the foreign key relationships within the underlying data model to manipulate the products related to an opportunity or the contacts associated with the opportunity.

Business objects are the basis for views, which are then grouped into 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 definition include the All Opportunity List view, Opportunity Detail - Contacts view, and Opportunity Detail - Products view. The key to all of these views is that the driving data 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.

NOTE: In the Siebel eBusiness Application, under the Contacts screen, in any of the Contact views, users can create identical contact records — that is, records with the same values for first name and last name even when connected to the server database. This is the application's intended functionality. The user keys were altered to allow users to create identical contact records because there may be contacts with the same first name and last name in a large company database. To prevent the same contact from being entered twice, run real-time data matching using the data quality module.

You 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.

Specifying Links for Associated Business Components

To include a new business component on a business object definition, add a business object component record as a child of the business object. You also need to identify the link property on the business object component.

Although it is optional, you should enter a value for the Link property on the business object component. 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.

You must define a link for a business object component under the following circumstances:

- 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.

Managing Unused Business Objects

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.

Creating and Modifying Business Components

A business component is a logical representation of one or more tables. The information stored in a business component is usually specific to a particular subject, such as a product, a contact, or an account. This information may or may not depend on any other business component. A business component can be included in one or more business object definitions.

Most of the data for a business component comes from one main base table. For example, the Opportunity business component is based on the S_OPTY table. Most data retrieved for the Opportunity business component comes from this table. However, data from other tables are also available through joins to those tables.

Many business components are based on the S_PARTY table, where most information does not come from the main base table but from joined tables. For example, the Account business component is based on the S_PARTY table but most of the data for the Account business component is stored on the joined table S_ORG_EXT.

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 configuration that is smaller, requires less memory, 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.

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.

Associating Business Component Fields With Tables

A business component field directly relates to a column from a table in the database or a calculated value. All fields making up a business component record contain entries from both single-value and multivalue field types. Each multivalue field references a multivalue link. You can also associate a join with a field to retrieve data from a joined table. The join must be implicit for the business component or explicitly defined in the business component.

Defining System Fields

System fields should not be explicitly defined for a business component. The ROW_ID, UPDATE, UPDATED_BY, CREATED, and CREATED_BY columns are automatically populated during the insert process.

You can refer to a system field on another business component, without the system field being explicitly defined on that business component. The system field becomes an option to choose from in the picklist of fields. They are listed as Id, Updated, Updated By, Created and Created By.

Constructing 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 none or only one record. For example, a join is used to retrieve the primary Account for an Opportunity.

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 in order to make sure that the explicit join does not get used instead. To make sure that no conflict exists, you should always give the join a distinct and custom alias name.

There are two types of joins, explicit and implicit.

Creating Explicit Joins

Explicit joins consist of the following:

- **Joins to Non-Party Related Tables**. In an explicit join, 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.
- **Joins to Party Related Tables**. Joins to tables that are extensions of the S_PARTY table (such as S_CONTACT, S_ORG_EXT, or S_USER, S_POSTN) require that the foreign key value be exposed as the source field, as stated previously. However, the destination Column must reference the PAR_ROW_ID column in the joined table.

Creating Implicit Joins

Implicit joins have the following properties:

- You do not have to define an implicit join through Siebel Tools. These joins exist for all 1:1 (_X) extension tables and relevant intersection tables.
- Implicit joins exist for party related entities. There is an implicit join available for each extension table belonging to a base table. S_PARTY has many extension tables, including S_ORG_EXT, S_CONTACT, S_POSTN, and S_USER. These implicit joins are used in other party business components to map their main data. For example, if you added a field to the Account business component and then selected the Join property, you would see several 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_USER. These are implicit joins. The Join aliases are displayed as options in the Join property pick list on the Field object.
- You can update the columns from this type of join.

Creating a New Join

You can create a new join in these circumstances:

- 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 on an existing join.
- When mapping fields in party business components, use the implicit join for the extension table.
- 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.

For more information, see *Siebel Tools Reference, MidMarket Edition*.

Constructing Links

Links specify the relationship between two business components. Like business objects, a link is a top-level object with no parent object. Both business objects and multi-value links in business components reference links.

Creating One-to-Many (1:M) Links

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 has 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.

Creating Many-to-Many (M:M) Links

In a many–to–many link, you define the child and parent business components. You also define the Inter Table. From the Inter Table, you define the Inter Child Column and the Inter Parent Column (the foreign keys to the child and parent, respectively). 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.

Setting Advanced Link Properties

When you create a link, you can set the following properties:

- Cascade Delete
- Search Specification
- Sort Specification
- Visibility Rule

Setting the Cascade Delete Property

The Cascade Delete property determines whether a child record is deleted when the parent record is deleted. If the property is set to Delete, then the child record is deleted. This is most appropriate for values stored on _XM tables where the only related record is the parent record. If the property is set to Clear, then the child record is not deleted. 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. The default value for Cascade Delete is None. With this setting, no records are deleted and the foreign key column is not cleared.

CAUTION: Take special precautions when determining this property. If set incorrectly, it may cause data integrity issues or orphaned records.

Setting the Search Specification Property

The Search Specification 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.

Setting the Sort Specification Property

This applies only to association lists.

Applying the Visibility Rule Property

This property determines whether visibility will be applied in the Link. The properties are Never, Always Child, and Drill Down.

For more information on these properties, see *Siebel Tools Online Help, MidMarket Edition*.

This section provides guidelines for configuring user interface objects.

Creating and Modifying User Interface Objects

User interface objects determine what a user sees and how a user interacts with the Siebel eBusiness Application. This section provides guidelines for configuring the following user interface objects:

- Screen
- View
- Applet
- Control
- List Column

Configuring Screens

A screen object is used to group or modify a collection of views.

Creating and Modifying Screens

Typically, you create a new screen when you create a new business object. You modify an existing screen object when you remove or add a screen tab, change menu text (for the Site Map), or define new views for that screen. Screens appear as first-level choices on the Site Map.

Views associated with screens appear in one of two places. Views with visibility properties (specifically the Visibility applet) appear in the Site Map as views accessible under the screen. They also appear in the Show drop–down list in the screen's default view. Screens are also accessed through screen tabs. Each screen tab represents a screen. The views that are part of that screen are available in the Show drop–down list (those with visibility properties) or are represented as view tabs under the screen's default view.

The order of the tabs and their listing on the Site Map is controlled by the sequence property of the records in the page tab and screen menu item child object of the application object.

Managing Unused Screens

If you will not use a standard screen in your implementation, use the Responsibility Administration Screen to disassociate all views on the redundant screen from those responsibilities your organization uses. This approach reduces the amount of configuration necessary for you to maintain and upgrade. 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—it will not be compiled in the SRF.

Configuring Views

Typically, a view defines a visual representation of a business object's data and is composed of a collection of applets. There can be many views of information for a business object. The following sections give guidelines for which views not to configure, naming views, handling unused views, and choosing a view layout.

Creating and Modifying Views

During the initial stages of an implementation, you define the required views. These views are usually in one of these categories:

- Views that are standard and do not need to be modified.
- Views that closely align with an existing view and require the same applet layout, but require other configurations such changing a title, inactivating or adding controls or list columns, and changing display labels. For this category, it is also recommended that you configure the existing view object. Most configurations in this category are actually done on the applets instead of the views.
- Views that closely align with an existing view but require a moderately different applet layout, possibly displaying applets based on new business components or adding toggles. For this category, it is also recommended that you configure the existing view object. The design may also require configuration of the existing view applets.
- Views that consolidate two already existing views. For this category, it is recommended that you configure one of the existing views by modifying the view object, and remove visibility to the redundant view using the Responsibility Administration screen.
- Views that do not have an obvious preexisting equivalent view. These are views that expose new functionality specifically configured for your implementation, exposing new business objects or business components. For this category, it is recommended that you create a new view object and avoid extensively modifying an existing definition that will not be used in your implementation. The resulting configurations will be much cleaner and easier to maintain and upgrade (both manually and automatically).

A view can be associated with more than one screen, but this configuration 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.

Modifying Server Administration Views

CAUTION: Do not modify Server Administration views. Information in these views is read from the siebens.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 siebens.dat file. However, it is not possible to configure the product to store such information in siebens.dat. Therefore, configuration of server views is not recommended or supported.

Calendar Views Not To Be Reconfigured

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

However, configuration of the eCalendar Detail View is fully supported through Siebel Tools.

Setting Interactivity Modes for Calendar Views

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 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 4.

Table 4. 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, MidMarket Edition*.)
- 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.

NOTE: 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.

Configuring Threads

Views associated with more than one screen in a given application will cause incorrect behavior in Siebel applications. When the thread is saved in the session file, the name of the view is saved without the name of the associated screen. When the end user chooses a thread that navigates to a duplicated view, Siebel applications will always navigate to one screen only—even if the thread was created in the other screen. Furthermore, if the duplicate view is defined as the default view on both tabs, the end user will see an anomaly in the user interface. Siebel applications will select one tab as the active tab when either of the tabs is selected. The duplicate tab will never appear to be active.

NOTE: The Thread Applet property must be correct, especially if a custom applet is placed in Sector 0. If this property is not set with the correct applet name, planned CTI screen pops for transfer calls will not work.

Managing Unused Views

If you are not using a standard view object, use the Responsibility List Administration screen to disassociate the redundant view from any responsibilities your organization uses. This offers an easy upgrade path if you decide to expose the view later. At that time, no configuration or software upgrade is required; you need only to reassign the view to the relevant responsibility. You can also inactivate the view within Siebel Tools.

Displaying View Titles

There are three different titles displayed for a view, as follows:

- **Title bar of the Siebel application window.** The title appears in the title bar, prefixed by the application name and a hyphen, as in Siebel Sales - Account List View. This is specified in the Title property of the view.
- **View bar tab.** In the View bar in the appropriate screen, the tab that navigates to this view. This is specified in the Viewbar Text property of the corresponding screen view object definition.
- **Screens menu sub-option.** In the Screens menu, as a sub-option of the appropriate screen, the menu option that navigates to this view. This is specified in the Menu Text property of the corresponding screen view object definition.

Keep these three title definitions consistent for one view. If at all possible, the text should be identical in all three.

If a view specifies a visibility mode, as indicated by a non-blank Visibility Applet Type property, the title (in all three locations) needs to identify the visibility mode, as indicated in Table 5.

Table 5. View Titles by Visibility Mode

Visibility Mode	Title Format	Example
Sales team visibility	My *buscomp*(s)	My Contacts
Personal visibility	My Personal *buscomp*(s)	My Personal Contacts
Manager visibility	My Team's *buscomp*(s)	My Team's Opportunities
All visibility	All *buscomp*(s)	All Accounts

For more information on visibility modes, refer to Chapter 9, "Visibility," and *Siebel Tools Reference, MidMarket Edition*.

Configuring Applets

An applet is composed of controls and occupies part of a view. You can configure an applet to allow data entry, provide a table of data rows, and display business graphics or a navigation tree. It provides viewing, entry, modification, and navigation capabilities for data in one business component. An applet is always associated with one and only one business component.

Creating and Modifying Applets

During the initial stages of an implementation, you define the required views. The applets displayed on these views usually fall into one of these categories:

- Applets that are standard and do require any modification.
- Applets that closely align with an existing list applet, but require minor configuration such as a title change, inactivating or adding controls or list columns, and changing display labels. For this category, it is recommended that you configure the existing applet object and its child objects.
- Applets that represent an already existing relationship (for example, opportunity contacts), but require extensive modification of the existing applet to produce a new applet layout. Modifications are considered extensive if the new applet requires a large combination of different configurations, such as resequencing of existing controls or list columns, inactivating and adding new controls and list columns. For this category, it is recommended that you create a new applet object (by copying an applet that closely resembles the applet you want to create), and then modify the copied applet. The resulting configuration will be much cleaner and easier to maintain and upgrade (both manually and automatically).
- You may need an applet with different drilldowns in different views.
- Applets that do not have an equivalent existing applet. These tend to be applets that expose a new business component. Always create a new applet for this category of list applet.

NOTE: Remember to set the Upgrade Ancestry property on all custom applets that are cloned from another applet.

Modify, rather than copy, an applet, unless you are making extensive modifications to the applet. This avoids having to change all references of that applet to the new copy.

The following are examples of situations in which you might need to copy an applet:

- When you must extensively modify an existing applet.
- When you need a Read Only copy of an existing applet.

NOTE: Do not change the Class property of preconfigured applets.

The objective of your design and configuration projects should be to produce a consistent and intuitive user interface. Wherever possible, applets displaying the same business component should be consistent across different screens and views. For example, the contact list displayed for an opportunity should be consistent with the contact list displayed for an account. Whenever possible, reuse applet definitions between different views and screens. Obvious exceptions include redundant controls or list columns and controls, or list columns that are relevant only to the current parent business component. For example, you would display the contact's account information when displaying opportunity contacts, but not when displaying account contacts. This recommendation does not necessarily apply when comparing list with form or entry applets. When the screen space for a view is limited, it may be practical to include fields at the end of a list applet that are not displayed on the associated entry applet.

Another approach to increasing the number of fields available in a form applet is to use applet toggling. You can define two applets based on the same business component to show information from the same record (but the field controls are distributed over multiple applets) and the user can toggle between the two.

Managing Unused Applets

It is recommended that you do not modify any applet you are not using in your implementation, or that you mark the applet as inactive.

Defining Online Help IDs

If you want to provide context-sensitive help in your application, you need to define Help IDs for screens and views. If you do not want context-sensitive help and you do not define a Help ID, the default start page for help will appear. For more information, see *Online Help Development Guide, MidMarket Edition*.

Displaying Applet Titles

The applet title is the value in the Title property. It determines what displays in the tab at the upper left of an applet in a view, or in the title bar of a pop-up applet. 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 6 offers standard conventions for the titles of certain applet types.

Table 6. 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

Configuring Controls and List Columns

The following sections list general guidelines for creating and modifying controls and list columns.

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).

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. The system fields are listed in the following table.

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.
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).

To define a control object

1. Create a new control object.
2. Enter a Control Name property of Last Updated where the text describes what you are attempting to expose.
3. Enter the Control Field property of *xxxx* where *xxxx* represents the system field you are attempting to expose.

Managing Unused Controls and List Columns

Set the Visible property to FALSE if the control or list column is Siebel-created, and delete if user-created.

Creating Text Controls and List Columns

Follow these guidelines when creating text controls or list columns:

- Use the appropriate pop-up device wherever possible to ease data entry. For instance, you should associate a calendar widget with a date field, and a multi-line edit box with a multi-line text field.
- Left-align data unless you have a specific reason to center or right-align it.

Creating Buttons

To create a new button, click the button icon in the Control toolbar and draw the button on the applet layout. You can alter the size of a button to accommodate a longer or shorter button label.

Defining Check Boxes

Follow these guidelines when defining check boxes:

- In form applets, position the label to the left of the check box. Right-align and vertically center the label.
- 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 text box labeled Shipping Method, with a list of values containing Standard and Next Day.
- Avoid negatives. For example, instead of *Not Required,* use *Optional*.

Referencing Controls and List Columns in Association Applets

All controls and list columns in an association applet must reference a corresponding field in the business component that is being associated. Examine the details of the Source Association Applet that is called from the Source MVG Applet on the Opportunity Entry Applet. All controls and list columns on the association applet reference a corresponding field on the Source business component.

Performance Guidelines 5

This section discusses the performance impact of multivalue links and reusing standard columns. In addition to this section, see also the appropriate sections in *Siebel Tools Reference, MidMarket Edition* for information on improving the performance of your configured Siebel application.

Multivalue Link Underlying Multivalue Groups

The way a multivalue link (MVL) underlying a multivalue group (MVG) is configured has a major impact on an application's performance. Whenever possible, configure an MVL underlying an MVG displayed in a list applet to use a primary join. To use a primary join, set the following attributes:

Attribute	Value
PrimaryIdField	< fieldname from parent business component >
UsePrimaryJoin	True

The field specified in the PrimaryIdField property stores the ROW_ID of the primary child record. If there is no child record, or the primary child record has not been defined, then the primary ID field stores the value as No Match Row Id.

The parent business component of a multi-value link is usually the same as the business component in which the MVL is defined. However, by using the SrcField property of the multi-value link object, you can create an MVL whose parent business component is related to the current business component indirectly through a join or through another MVL.

Indirect Multivalue Links

Every multivalue link in Siebel eBusiness applications is based on an underlying link object, whose name is specified by the MVL's DestLink property. Every link, in turn, defines a one-to-many or many-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 multivalue link in the Account business component:

```
[Multivalue link]
   DestBusComp = Business Address
   DestLink = Account/Business Address
   PrimaryIdField = Primary Address Id
   CheckNoMatch = TRUE
   PopupUpdOnly = TRUE
   NoCopy = TRUE
```

The DestLink property indicates that this MVL is based on the Account/Business Adress link, which is itself defined (in Link.odf) as:

```
[Link]
   Name = Account/Business Address
   ParentBusComp = Account
   ChildBusComp = Business Address
   DestField = Account Id
   CascadeDelete = Delete
```

Note that the ParentBusComp of this link is the Account business component, which is also the business component in which the MVL has been defined. So, in this typical MVL configuration, the multi-value group is populated with all of the children Business Address records for whichever account is currently selected in the Account business component.

Indirect Multivalue Links Through Joins

Although the parent business component of a multi-value group (MVG) is usually the same as the business component in which the MVL is defined, this is not always the case. For example, the Opportunity business component, like the Account business component, contains a multi-value group of business addresses. In this case, the business addresses are not directly related to the opportunities themselves; instead, they are children records of whichever account is associated with the current opportunity (if there is such an account). In order to correctly populate this MVG, Siebel eBusiness Applications need to know how to find the appropriate parent account record for this link, given the current record in the Opportunity business component. The SrcField property of the multivalue link object exists for this purpose.

Using the preceding example, the Business Address MVL is defined as follows within the Opportunity business component:

```
[Multivalue link]

   SrcField = Joined Account Id

   DestBusComp = Business Address

   DestLink = Account/Business Address

   PrimaryIdField = Primary Address Id

   CheckNoMatch = TRUE

   PopupUpdOnly = TRUE
```

Note that the DestLink of this MVL is still the Account/Business Address link, which defines the one-to-many relationship between accounts and business addresses. So, in this case, the parent business component of the link (that is, Account) is not the same as the business component in which the MVL is defined (that is, Opportunity). To determine the appropriate Account records for which to get the children Business addresses, Siebel eBusiness Applications look at the SrcField property of the MVL. For this particular multi-value link, the SrcField property refers to the Joined Account Id field in the Opportunity business component, which maps to the ROW_ID database column from the joined S_ORG_EXT table. So, for each Opportunity record, Siebel eBusiness Applications will populate the MVG with all Business Address records that are children of whichever account is indicated by the ROW_ID value stored in the Joined Account Id field.

Nested Multivalue Links

As previously described, a multivalue group can be populated through a joined table by using the SrcField property to refer to a business component field that is a foreign key to the joined table. Although less common, it is also possible to populate a multivalue group through another multivalue link from the same business component by using the SrcField property to refer to a multivalue field instead of a simple (single value) field. For example, consider the relationship between employees, positions, and territories. In Siebel eBusiness applications, one employee can hold multiple positions, while each position can be assigned to multiple territories. This means there is an Employee/Position link that defines the many-to-many relationship between employees and positions and there is a Position/Territory link that defines the many-to-many relationship between positions and territories.

The Employee business component (in Employee.odf) includes multivalue links for both positions and territories. The Position MVL is a *standard* one (that is, without a SrcField property):

```
[Multivalue link]
   DestBusComp = Position
   DestLink = Employee/Position
   PrimaryIdField = Primary Position Id
   NoInsert = TRUE
   UsePrimaryJoin = FALSE
```

However, because territories and employees are related only indirectly through positions, the Territory MVL is actually based on the Position/Territory link and must reference Position Id as its SrcField to find the appropriate position to use as the parent record for this link:

```
[Multivalue link]
   SrcField = Position Id
   DestBusComp = Territory
   DestLink = Position/Territory
   NoInsert = TRUE
```

Each employee can hold multiple positions, so the Position Id field is itself defined as a multivalue field (from the Position MVG) in the Employee business component:

```
[Multivalue link]
   Name = Position Id
   Field = Id
   MultiValueLink = Position
   ReadOnly = TRUE
```

In other words, the Position MVG in the Employee business component shows all of the child Position records for the current employee, while the Territory MVG in this same business component shows all of the child Territory records for whichever Position record is currently selected in the Position MVG.

This type of nested multivalue group may have significant performance implications because of the extra subqueries that must be executed. Furthermore, it is often unclear to the user that changing the currently selected value in one MVG causes another MVG to display an entirely new set of records. Because of these performance and usability concerns, such nested MVGs appear very rarely in the standard configurations of Siebel eBusiness Applications. You should generally avoid them in custom configurations.

Auto Primary

The AutoPrimary property determines the setting of a primary child record for a given parent record. If necessary, the Auto Primary property also determines the default value of the primary selection. The possible values for Auto Primary are as follows:

- **DEFAULT.** The first record automatically becomes the primary.
- **NONE.** You must manually specify the primary using the MVG applet.
- **SELECTED.** Selecting a primary on one MVL causes the selection of a primary on the others. For example, as soon as a primary Shipping Address is indicated, it also becomes the primary Billing Address.

SELECTED applies only when there are several multivalue links from one business component that all point to the same detail business component. This is the case for the Bill To Business Address and Ship To Business Address multivalue links in a standard Siebel Sales application. These multivalue links exist under both the Quote and Account business components. In this case, an example of the desired behavior is as follows:

- If you do not set a primary for the Bill To address, then do a separate query to bring back all addresses associated with the account (or order), and check to see whether one of the addresses was selected as primary for the Ship To address. If an address is already selected as primary, the SELECT (that is, set) is used and that address is also used as the primary Bill To address.

CheckNoMatch

The property CheckNoMatch may be set to TRUE or FALSE. This property controls the application's behavior when an MVL uses a primary join, but the primary ID field has the value No Match Row Id.

Typically, when a multivalue link has been configured with a primary join, the foreign key used by this join to identify the primary record may not match the primary record. For example, this can happen when the primary record has been deleted from the multivalue group or the multivalue group is new and has no records. In these cases, you can configure the multivalue link to update the primary foreign key to a value of NULL, or to a special value of NoMatchRowId, depending on your requirements. You configure this behavior through the Check No Match property of the Multi Value Link object type; however, there are consequences for the application's performance. The special NoMatchRowId value is designed to prevent secondary queries on foreign key values that are known to have failed, which improves performance the same way using a primary join improves performance.

You activate the NoMatchRowId generating and subsequent testing of behavior by setting Check No Match to FALSE for the MVL. This setting has the following results:

- When you have a master record where the primary foreign key is NULL or invalid, do a secondary query to determine if there are detail records in the multi-value group. If there are no detail records, set the primary ID field to the special value NoMatchRowId.
- When you have a master record where the primary foreign key has the value NoMatchRowId, this tells the system that there are no detail records in the multi-value group and the secondary query is not performed. This is not a permanent setting—you can update the primary after it is set to NoMatchRowId.

When this property is set to TRUE, check to see that there really are no child records. Do this check by executing a query against the child business component. If, in this case, the Auto Primary property is set to Default, the first record returned is set as the primary. If the Auto Primary property is set to SELECTED, check to determine whether any other multi-value link to this business component has indicated a primary, and set that record as the primary of this multi-value link.

AutoPrimary, CheckNoMatch and Performance

The combined settings of either of the following attributes and values will cause reduced performance in an environment with a high number of parent records without primary child records, as the application is forced to check the existence of child records by executing secondary queries. Both settings of the AutoPrimary property allow the user to set only the primary child records, so the reduced performance could have a long duration.

Attribute	Value
AutoPrimary	Selected
CheckNoMatch	True

Attribute	Value
AutoPrimary	None
CheckNoMatch	True

Consider using the same environment with a high number of parent records without primary child records and the following settings:

Attribute	Value
AutoPrimary	Default
CheckNoMatch	True

This configuration could also cause poor performance when displaying the parent records for the first time. Setting CheckNoMatch forces the application to execute a second query to check the existence of the child records. Because of the AutoPrimary property setting, the application sets the first read child record as the primary. This way, the parent records get a primary child record, and the performance significantly improves the next time these parent records are displayed.

Set Check No Match to FALSE for most multi-value links because of the performance consequences. Set it to TRUE only if the multi-value group could possibly have records added to it without going through the multi-value group itself. For example, account addresses might actually be inserted through the Business Address multi-value group on the Contact business component instead of the Business Address multi-value group on the Account business component. Also, if records can be added to the detail business component through Enterprise Integration Manager, the TRUE setting is the appropriate one.

Reusing Standard Columns

The architecture and data model of your application has been tuned for best performance. This optimization is achieved by using proper indexes, data caching, efficient SQL generation and also by denormalizing columns on certain tables. These denormalized columns are indexed so that the application can improve the performance of complex SQL statements by using these columns for search or sort operations instead of the columns of the original table.

This is why you should not change the use of some columns, such as NAME, LOC of the S_ORG_EXT and LAST_NAME, FST_NAME, MID_NAME of S_CONTACT.

Table S_ORG_EXT: Reusing NAME and LOC

The columns NAME and LOC of the S_ORG_EXT are denormalized into ACCNT_NAME and ACCNT_LOC in the S_ACCNT_POSTN table. When sorting the Accounts by name and location in views with the property VisibilityAppletType = Sales Rep, the application uses the denormalized columns ACCNT_NAME and ACCNT_LOC of the S_ACCNT_POSTN table. This choice allows the use of an index.

If the account name and location were stored in extension columns (for example, X_NAME and X_LOC), these columns would have to be used for sorting instead of NAME and LOC. Even if these extension columns were indexed, the application could not use an existing index to create the necessary joins and sort the data, because the index is on S_ORG_EXT and not on S_ACCNT_POSTN. Therefore, the result would be a significant decrease in performance, as shown in the following example.

The first SQL statement is generated by the standard My Accounts view. The query plan shows that the database uses numerous indexes to execute the statement.

```
SELECT

      T1.LAST_UPD_BY,

      T1.ROW_ID,

      T1.CONFLICT_ID,

      .

      .
```

```
      .

      T10.PR_EMP_ID,

      T2.DUNS_NUM,

      T2.HIST_SLS_EXCH_DT,

      T2.ASGN_USR_EXCLD_FLG,

T2.PTNTL_SLS_CURCY_CD,

T2.PAR_OU_ID

   FROM

      SIEBEL.S_PARTY T1

         INNER JOIN SIEBEL.S_ORG_EXT T2 ON T1.ROW_ID = T2.PAR_ROW_ID

         INNER JOIN SIEBEL.S_ACCNT_POSTN T3 ON (T3.POSITION_ID = ?,
         0.05)

AND T2.ROW_ID = T3.OU_EXT_ID

         INNER JOIN SIEBEL.S_PARTY T4 ON (T4.ROW_ID = T3.POSITION_ID,
         0.05)

         LEFT OUTER JOIN SIEBEL.S_PRI_LST T5 ON T2.CURR_PRI_LST_ID =
         T5.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_INVLOC T6 ON T2.PR_FULFL_INVLOC_ID
         =

T6.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_ORG_EXT T7 ON T2.PAR_OU_ID =
         T7.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_ORG_EXT_SS T8 ON T1.ROW_ID =
         T8.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_INT_INSTANCE T9 ON T8.OWN_INST_ID =

T9.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_POSTN T10 ON T2.PR_POSTN_ID =
         T10.PAR_ROW_ID
```

```
        LEFT OUTER JOIN SIEBEL.S_USER T11 ON T10.PR_EMP_ID =
        T11.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ADDR_ORG T12 ON T2.PR_ADDR_ID =
        T12.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_INDUST T13 ON T2.PR_INDUST_ID =
        T13.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ASGN_GRP T14 ON T2.PR_TERR_ID =
        T14.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_POSTN T15 ON T3.POSITION_ID =
        T15.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_USER T16 ON T15.PR_EMP_ID =
        T16.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_SYN T17 ON T2.PR_SYN_ID =
        T17.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_BU T18 ON T2.BU_ID = T18.BU_ID
        AND

T2.ROW_ID = T18.ORG_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T19 ON T18.BU_ID = T19.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT T20 ON T18.BU_ID =
        T20.PAR_ROW_ID

     WHERE

        ((T2.INT_ORG_FLG != 'Y' OR T2.PRTNR_FLG != 'N') AND
        (T3.ACCNT_NAME >= ?))

   ORDER BY

        T3.POSITION_ID, T3.ACCNT_NAME

Query plan :
T3(S_ACCNT_POSTN_M1),T2(S_ORG_EXT_P1),T1(S_PARTY_P1),T15(S_POSTN_U
2),T10(S_POSTN_U2),T4(S_PARTY_P1),T12(S_ADDR_ORD_P1),T13(S_INDUST_
P1),T7(S_ORG_EXT_U3),T16(S_USER_U2),T11(S_USER_U2),T17(S_ORG_SYN_P
1),T6(S_INVLOC_P1),T5(S_PRI_LST_P1),T14(S_ASGN_GRP_P1),T18(S_ORG_B
U_U1),T19(S_PARTY_P1),T20(S_ORG_EXT_U3),T8(S_ORG_EXT_SS_U1),T9(se)
```

The second of the following SQL statements has a different ORDER BY clause. Even though the columns NAME and LOC of S_ORG_EXT are indexed, the database cannot use this index. Performance decreases from the use of a temporary table. The same behavior occurs if the ORDER BY clause uses the columns X_NAME and X_LOC instead of NAME and LOC.

```
SELECT

      T1.LAST_UPD_BY,

      T1.ROW_ID,

      T1.CONFLICT_ID,

      .

      .

      .

      T10.PR_EMP_ID,

      T2.DUNS_NUM,

      T2.HIST_SLS_EXCH_DT,

      T2.ASGN_USR_EXCLD_FLG,

      T2.PTNTL_SLS_CURCY_CD,

      T2.PAR_OU_ID

   FROM

      SIEBEL.S_PARTY T1

         INNER JOIN SIEBEL.S_ORG_EXT T2 ON T1.ROW_ID = T2.PAR_ROW_ID

         INNER JOIN SIEBEL.S_ACCNT_POSTN T3 ON (T3.POSITION_ID = ?,
         0.05) AND

T2.ROW_ID = T3.OU_EXT_ID

         INNER JOIN SIEBEL.S_PARTY T4 ON (T4.ROW_ID = T3.POSITION_ID,
         0.05)

         LEFT OUTER JOIN SIEBEL.S_PRI_LST T5 ON T2.CURR_PRI_LST_ID =
         T5.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_INVLOC T6 ON T2.PR_FULFL_INVLOC_ID
         =
```

```
T6.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT T7 ON T2.PAR_OU_ID =
        T7.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT_SS T8 ON T1.ROW_ID =
        T8.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_INT_INSTANCE T9 ON T8.OWN_INST_ID =

T9.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_POSTN T10 ON T2.PR_POSTN_ID =
        T10.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_USER T11 ON T10.PR_EMP_ID =
        T11.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ADDR_ORG T12 ON T2.PR_ADDR_ID =
        T12.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_INDUST T13 ON T2.PR_INDUST_ID =
        T13.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ASGN_GRP T14 ON T2.PR_TERR_ID =
        T14.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_POSTN T15 ON T3.POSITION_ID =
        T15.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_USER T16 ON T15.PR_EMP_ID =
        T16.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_SYN T17 ON T2.PR_SYN_ID =
        T17.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_BU T18 ON T2.BU_ID = T18.BU_ID
        AND

T2.ROW_ID = T18.ORG_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T19 ON T18.BU_ID = T19.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT T20 ON T18.BU_ID =
        T20.PAR_ROW_ID

     WHERE

        ((T2.INT_ORG_FLG != 'Y' OR T2.PRTNR_FLG != 'N') AND

        (T3.ACCNT_NAME >= ?))
```

```
   ORDER BY

        T3.ACCNT_NAME, T3.POSITION_ID

Query plan : TEMPORARY TABLE
T3(S_ACCNT_POSTN_M1),T2(S_ORG_EXT_P1),T1(S_PARTY_P1),T15(S_POSTN_U
2),T10(S_POSTN_U2),T4(S_PARTY_P1),T12(S_ADDR_ORG_P1),T13(S_INDUST_
P1),T7(S_ORG_EXT_U3),T16(S_USER_U2),T11(S_USER_U2),T17(S_ORG_SYN_P
1),T6(S_INVLOC_P1),T5(S_PRI_LST_P1),T14(S_ASGN_GRP_P1),T18(S_ORG_B
U_U1),T19(S_PARTY_P1),T20(S_ORG_EXT_U3),T8(S_ORG_EXT_SS_U1),T9(se)
```

Table S_ CONTACT: Reusing LAST_NAME, FST_NAME, MID_NAME

The columns LAST_NAME, FST_NAME, MID_NAME of the S_CONTACT table are denormalized into three other columns in the S_POSTN_CON table: CON_LAST_NAME, CON_FST_NAME, CON_MID_NAME.

The visibility setting of the view often governs the sorting of data in the view. A good example of this is when *VisibilityAppletType = Sales Rep*, which is the case on the My Contacts view.

Here, the application uses the denormalized columns CON_LAST_NAME and CON_FST_NAME of the S_POSTN_CON table. This forces the database to use a particular index.

If the contact name was stored in extension columns in the S_CONTACT table (for example, X_LAST_NAME and X_FST_NAME), these columns would be used for sorting (as a result of existing sort specifications, user interaction, and querying) instead of the standard LAST_NAME and FST_NAME columns.

Even if these extension columns were indexed, the application could not use the index for creating the needed joins and for sorting the data, because the index is on S_CONTACT and not on S_POSTN_CON. This would result in a significant reduction in performance, especially with larger sets of results, as shown in this example.

The first SQL statement below is generated by the standard view My Contacts. The query plan shows that the database uses indexes to execute the statement.

```
SELECT

      T1.LAST_UPD_BY,

      T1.ROW_ID,

      T1.CONFLICT_ID,

      .

      .

      .

      T15.LAST_UPD_BY,

      T2.CON_MANAGER_PER_ID,

      T16.CITY,

      T2.ASGN_USR_EXCLD_FLG

   FROM

      SIEBEL.S_PARTY T1

         INNER JOIN SIEBEL.S_CONTACT T2 ON T1.ROW_ID = T2.PAR_ROW_ID

         INNER JOIN SIEBEL.S_POSTN_CON T3 ON (T3.POSTN_ID = '1-C1RC',
         0.05)

AND T2.ROW_ID = T3.CON_ID

         INNER JOIN SIEBEL.S_PARTY T4 ON (T4.ROW_ID = T3.POSTN_ID,
         0.05)

         LEFT OUTER JOIN SIEBEL.S_BU T5 ON T2.BU_ID = T5.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_USER T6 ON T2.EMP_ID =
         T6.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_CONTACT T7 ON
         T2.CON_MANAGER_PER_ID=

T7.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_ORG_EXT T8 ON T2.PR_DEPT_OU_ID =
```

```
T8.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_TIMEZONE T9 ON T2.TIMEZONE_ID =
        T9.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_CONTACT_SS T10 ON T1.ROW_ID =
        T10.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_INT_INSTANCE T11 ON T10.OWN_INST_ID
        =

T11.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_TIMEZONE_LANG T12 ON T9.ROW_ID =

T12.PAR_ROW_ID AND T12.LANG_ID = 'ENU'

        LEFT OUTER JOIN SIEBEL.S_EMP_PER T13 ON T1.ROW_ID =
        T13.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_CONTACT_X T14 ON T1.ROW_ID =
        T14.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_PER_PRTNRAPPL T15 ON T1.ROW_ID =

T15.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ADDR_ORG T16 ON T2.PR_OU_ADDR_ID =

T16.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T17 ON T2.PR_DEPT_OU_ID =
        T17.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT T18 ON T2.PR_DEPT_OU_ID =

T18.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_POSTN T19 ON T3.POSTN_ID =
        T19.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_USER T20 ON T19.PR_EMP_ID =
        T20.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T21 ON T2.PR_GRP_OU_ID =
        T21.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_GROUP T22 ON T2.PR_GRP_OU_ID =
```

```
T22.PAR_ROW_ID

     WHERE

        ((T2.PRIV_FLG = 'N') AND

        (T3.CON_LAST_NAME >= ?))

     ORDER BY

        T3.POSTN_ID, T3.CON_LAST_NAME, T3.CON_FST_NAME

Query plan :
T3(S_POSTN_CON_M1),T2(S_CONTACT_P1),T1(S_PARTY_P1),T1(S_PARTY_P1),
T19(S_POSTN_U2),T9(S_TIMEZONE_P1),T21(S_PARTY_P1),T17(S_PARTY_P1),
T4(S_PARTY_P1),T7(S_CONTACT_U2),T16(S_ADDR_ORG_P1),T18(S_ORG_EXT_U
3),T8(S_ORG_EXT_U3),T13(S_EMP_PER_U1),T20(S_USER_U2),T6(S_USER_U2)
,T22(S_ORG_GROUP_U2),T5(S_BU_U2),T10(S_CONTACT_SS_U1),T11(seq),T14
(S_CONTACT_X_U1),T12(S_TIMEZONE_LANG_U1),T15(S_PER_PRTNRAPPL_U1)
```

In the second SQL statement, the ORDER BY clause has been changed. Even though the columns LAST_NAME, FST_NAME of S_CONTACT are indexed, the database cannot use this index. Performance decreases from the use of a temporary table. The same behavior happens if the ORDER BY clause uses the columns X_LAST_NAME and X_FST_NAME instead of LAST_NAME and FST_NAME.

```
SELECT

     T1.LAST_UPD_BY,

     T1.ROW_ID,

     T1.CONFLICT_ID,

     .

     .

     .

     T15.LAST_UPD_BY,

     T2.CON_MANAGER_PER_ID,

     T16.CITY,

     T2.ASGN_USR_EXCLD_FLG
```

```
   FROM

      SIEBEL.S_PARTY T1

         INNER JOIN SIEBEL.S_CONTACT T2 ON T1.ROW_ID = T2.PAR_ROW_ID

         INNER JOIN SIEBEL.S_POSTN_CON T3 ON (T3.POSTN_ID = '1-C1RC',
         0.05)

AND T2.ROW_ID = T3.CON_ID

         INNER JOIN SIEBEL.S_PARTY T4 ON (T4.ROW_ID = T3.POSTN_ID,
         0.05)

         LEFT OUTER JOIN SIEBEL.S_BU T5 ON T2.BU_ID = T5.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_USER T6 ON T2.EMP_ID =
         T6.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_CONTACT T7 ON T2.CON_MANAGER_PER_ID
         =

T7.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_ORG_EXT T8 ON T2.PR_DEPT_OU_ID =

T8.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_TIMEZONE T9 ON T2.TIMEZONE_ID =
         T9.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_CONTACT_SS T10 ON T1.ROW_ID =
         T10.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_INT_INSTANCE T11 ON T10.OWN_INST_ID
         =

T11.ROW_ID

         LEFT OUTER JOIN SIEBEL.S_TIMEZONE_LANG T12 ON T9.ROW_ID =

T12.PAR_ROW_ID AND T12.LANG_ID = 'ENU'

         LEFT OUTER JOIN SIEBEL.S_EMP_PER T13 ON T1.ROW_ID =
         T13.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_CONTACT_X T14 ON T1.ROW_ID =
         T14.PAR_ROW_ID

         LEFT OUTER JOIN SIEBEL.S_PER_PRTNRAPPL T15 ON T1.ROW_ID =
```

```
T15.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ADDR_ORG T16 ON T2.PR_OU_ADDR_ID =

T16.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T17 ON T2.PR_DEPT_OU_ID =
        T17.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_EXT T18 ON T2.PR_DEPT_OU_ID =

T18.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_POSTN T19 ON T3.POSTN_ID =
        T19.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_USER T20 ON T19.PR_EMP_ID =
        T20.PAR_ROW_ID

        LEFT OUTER JOIN SIEBEL.S_PARTY T21 ON T2.PR_GRP_OU_ID =
        T21.ROW_ID

        LEFT OUTER JOIN SIEBEL.S_ORG_GROUP T22 ON T2.PR_GRP_OU_ID =

T22.PAR_ROW_ID

     WHERE

        ((T2.PRIV_FLG = 'N') AND

        (T3.CON_LAST_NAME >= ?))

     ORDER BY

        T3.CON_LAST_NAME, T3.CON_FST_NAME, T3.POSTN_ID

Query plan: TEMPORARY TABLE
T3(S_POSTN_CON_M1),T2(S_CONTACT_P1),T1(S_PARTY_P1),T19(S_POSTN_U2)
,T9(S_TIMEZONE_P1),T21(S_PARTY_P1),T17(S_PARTY_P1),T4(S_PARTY_P1),
T7(S_CONTACT_U2),T16(S_ADDR_ORG_P1),T18(S_ORG_EXT_U3),T8(S_ORG_EXT
_U3),T13(S_EMP_PER_U1),T20(S_USER_U2),T6(S_USER_U2),T22(S_ORG_GROU
P_U2),T5(S_BU_U2),T10(S_CONTACT_SS_U1),T11(seq),T13(S_CONTACT_X_U1
),T12(S_TIMEZONE_LANG_U1),T15(S_PER_PRTNRAPPL_U1)
```

NOTE: Do not remap existing fields, especially those based on User Key columns, to other columns in the same table.

Generating and Analyzing SQL

After configuring a view, you should always check the SQL that is generated during runtime. Use the `/s filename` parameter on the command line, as shown in the following example, to start the Siebel eBusiness application. This spools the generated SQL to a file. If you do not specify a path, the file is created in the Siebel root bin directory `c:\siebel\bin`. This file has all of the unique SQL statements generated during the current session and is overwritten during every new session.

The following example shows a command line using the `/s filename` parameter:

```
D:\siebel\core6\BIN\siebel.exe /c "D:\siebel\core6\bin\uagent.cfg" /s
"c:\siebel.doc"
```

NOTE: The spool file is simply a text file holding spooled SQL. Spooling it into a DOC file correctly formats it for viewing.

Once generated, analyze the contents of the file to identify any possible performance issues. Key indicators are:

- Complexity of SQL statements
- Order By Clause
- Where Clause
- Joins

Next, execute potentially problematic queries directly against the database and then generate a Query Plan. Use this plan to determine whether:

- Indexes are being used
- Tables Scans are occurring
- Temporary Tables are being generated

Finally, comparing with a standard application lets you identify any potentially slow queries.

You can resolve many performance issues either by modifying the business component or an applet's search specification and the business component's sort specification, or by creating new indexes on the base table. Only specially trained Siebel Systems personnel can modify existing Siebel indexes. This restriction is enforced so that performance in other modules (such as the Enterprise Integration Manager) is not adversely affected by any index modifications you make to improve query performance through the user interface.

Consider any potential performance implications before modifying the search specification and sort specification properties for a business component. By spooling out the SQL, as previously described, you can analyze which indexes are likely to be used when your application queries the business component, through each applet.

Also, use your Relational Database Management System (RDBMS) vendor's tools to analyze the SQL your implementation generated.

SQL Queries Against Database Data

The database that underlies Siebel applications can be queried to obtain information on a read-only basis. However, update queries should never be performed on the database. All data manipulation and restructuring should be performed through Siebel Tools or an end-user Siebel application such as Siebel Sales.

Development Standards for Siebel Script Languages and Object Interfaces

6

This section gives development standards for Siebel script languages and object interfaces.

Siebel Script Languages and Object Interfaces

Siebel eBusiness Applications allows functionality that cannot be implemented with declarative configuration to be developed with server script in eScript or Siebel Visual Basic (VB), or browser script in JavaScript. Generally, these script languages are used to complement existing configurable capabilities by allowing developers to fill in functionality gaps with concise, simple scripts.

In Siebel 7, script languages are divided into two types: server script and browser script. Server script is written in either eScript or Siebel VB and runs within the object manager on the Siebel Server or the Dedicated Web Client. Browser script is written only in JavaScript and executes within the browser on the client machine. Browser script supports a subset of the object interfaces available to server script. In most cases, browser script is used to augment server script to improve performance (by running logic on the client instead of the server), or to interact directly with the client machine.

In addition to the scripting languages there are several object interfaces available to external programs and programming languages in Siebel 7:

- COM Data Control
- COM Data Server
- Mobile or Dedicated Web Client Automation Server
- Siebel Smart Web Client Automation Server
- Java Data Bean
- XML Web Interface
- CORBA Interface

A detailed discussion of these object interfaces is beyond the scope of this guide. For more information, see *Siebel Tools Online Help, MidMarket Edition*. This guide applies to object interfaces that expose a programmatic interface (for example, access to Siebel objects) from within another programming language, such as Java, C++, or Visual Basic. The following object interfaces refer to all of the preceding interfaces except the XML Web Interface.

Preimplementation Considerations

Siebel scripting languages can add extensions to the existing declarative configuration capabilities. Along with this additional capability, there are also disadvantages, such as degraded maintainability, compromised upgrade options, and increased debugging time. Generally, you should try all other possibilities before using script to accomplish a functional requirement.

- During the initial high-level design, determine if the desired functionality is already part of the Siebel application or is available as a separate Siebel application module. The Siebel product suite is broad, and already includes a variety of business requirements as business components, business services, or separate products. Sometimes, the solution may be only a small piece of functionality (for example, visibility settings, user properties, search specifications on links, or specialized behavior of the base class) or it may be entire application modules (such as Forecasting). In either case, you should always use standard Siebel features instead of building a custom solution.

  The future benefits from using existing product functionality outweigh the additional licensing costs you may incur for using additional Siebel modules. Development costs for maintaining, upgrading, and debugging code can be considerable. Using or augmenting existing product behavior leaves the majority of these issues to Siebel engineering.

- Use existing business services for modular, generic behavior.

  There are over 300 prebuilt business services available, most of which can be used by scripts, workflows, or external programs. These business services provide data transformation, communication, server requests, and many other common functions. Using these business services either individually or in conjunction with each other can help avoid duplicating your development efforts.

- Use declarative configuration mechanisms whenever possible.

  There are many ways to configure different types of functionality in Siebel eBusiness Applications. Though they are not immediately obvious, most types of functional behavior are possible through declarative configuration.

  Table 7 lists the different ways to achieve application behavior that is often implemented in script.

Table 7. Available Declarative Configuration Options

Technology	Documentation
User Properties	*Siebel Tools Reference, MidMarket Edition*
Runtime Events	*Siebel Business Process Designer Administration Guide, MidMarket Edition*
Workflow	*Siebel Business Process Designer Administration Guide, MidMarket Edition*
State Model	*Siebel Business Process Designer Administration Guide, MidMarket Edition*
Visibility	*Siebel Tools Reference, MidMarket Edition*

The technologies in Table 7 perform a wide variety of tasks, such as simple data validation, event-driven processing, data-driven read-only behavior, interaction with external systems, security, or state transitions.

The preceding guidelines describe the two factors that have the most significant impact on application quality, maintenance and upgrades. Using standard Siebel components and declarative configuration tools decreases the amount of time and effort required to maintain the application. They also make it easier to get a high-level view of the type of functionality implemented, because all the functionality is stored in one place. Scripts require more research because you must examine each object individually and read script code to determine how it works.

In addition, declarative functionality is upgraded automatically when the application is upgraded, while scripts are not. After an upgrade is complete, you must test all objects that have scripts to make sure functionality has not been lost. You must also review all nontrivial scripts to make sure they are compatible with any changes in the application. Configuration mechanisms are rarely removed from the application, so most standard functionality will upgrade without incident. For information on whether an upgrade will affect declarative functionality, see the release notes documentation for your Siebel application.

Script Guidelines

The following sections give guidelines for the Siebel script languages. Because all of these guidelines will not apply to both server and browser script, they have been divided into sections. The first section applies to the two types of server script, eScript and Siebel Visual Basic (VB). The second section applies to browser script. Each guideline follows a similar format:

- Recommendations
- Explanations and examples
- Implications of not using the recommendations

As some issues apply only to eScript, Siebel VB, or the object interfaces, the applicable technologies will appear in parentheses.

Server Script and Object Interfaces

- Destroy object variables when they are no longer needed (eScript or Siebel VB).

  Set variables that represent Siebel objects explicitly to null (eScript) or nothing (Siebel VB) when they are no longer needed. This tells Siebel Object Manager that the object is no longer needed and that its reference counter can be decremented. If the object is no longer referenced anywhere else you can safely destroy it to free up memory and processor resources.

- Destroy variables that refer to the following object types:
  - Business components
  - Business objects
  - Business services
  - Property sets
  - Applets

- Destroy business objects and business components in the opposite order in which they are created. For example:

```
var boQuote = TheApplication().GetBusObject( "Quote" );

var bcQuote = boQuote.GetBusComp( "Quote" );

var bcQuoteItem = boQuote.GetBusComp( "Quote Item" );

//...

bcQuoteItem = null;

bcQuote = null;

boQuote = null;
```

  Unless you explicitly destroy object variables, memory and processor resources will not be made available. This negatively impacts the application's performance, especially if multiple clients are connected to an object manager, such as in the Siebel Smart Web Client. It can also affect the application's overall stability.

- Use the ForwardOnly cursor mode for ExecuteQuery() unless ForwardBackward is required (eScript, Siebel VB, or Object Interfaces).

  When using the ExecuteQuery() method to query a business component, explicitly specify the ForwardOnly cursor mode unless bidirectional iteration is required in the result set or the target business component is the basis for any user interface objects. When no argument to ExecuteQuery() is specified, the cursor mode defaults to ForwardBackward. For example:

```
bcBusAddr.ClearToQuery();

bcBusAddr.ActivateField( "Street Address" );

bcBusAddr.ActivateField( "City" );

bcBusAddr.ActivateField( "State" );

bcBusAddr.ExecuteQuery( ForwardOnly );


if ( bcBusAddr.FirstRecord() ) {

   // ...
```

In this example, the code can safely iterate from the first record to the last using the FirstRecord() and NextRecord() methods.

When ForwardBackward mode is used, all records retrieved in the result set must be cached on the host machine until the object is destroyed or requeried. In other words, if the above code used ForwardBackward mode instead and iterate to the last record returned, all records would remain locally cached the entire time the record set is being used. Because of this, code that moves backward with PreviousRecord() or FirstRecord() (this is only moving backward when the current record is not the first record in the result set) does not have to refresh data from the database server, so response time is optimal.

However, if backward iteration is not necessary, then the entire result set is retained for no reason. Memory occupied by records that are no longer needed is periodically released in ForwardOnly mode. In applications where multiple clients are active on an Object Manager, this technique uses a significant amount of memory, which can degrade the application's overall performance.

- Know when to explicitly activate fields (eScript, Siebel VB, or Object Interfaces).

  The ActivateField() business component method adds a field's column to the subsequent SELECT statement that will be generated the next time a business component is queried using ExecuteQuery(). In general, this method should be used as follows:

```
bcAccount.ClearToQuery();

bcAccount.ActivateField( "Name" );

bcAccount.ActivateField( "Location" );

bcAccount.ActivateField( "CSN" );

bcAccount.ExecuteQuery( ForwardOnly );


// ...


bcAccount.SetFieldValue( "CSN", "123" );

bcAccount.WriteRecord();
```

A common error is to activate fields after invoking ExecuteQuery() but before calling SetFieldValue(). This can cause mobile client synchronization to fail. Another common error is to explicitly activate system fields, which is unnecessary because they are always active. The following system fields are always active:

- Id
- Created
- Created By
- Updated
- Updated By

The following field attributes may force a field to always be active:

- If a currently displayed applet is based on the instance of a business component and the field is displayed in the applet, then it is active in that business component instance.
- If the field's Link Specification property is set to TRUE.
- If the field's Force Active property is set to TRUE.
- If the business component's Force Active property is set to TRUE.

Because the ActivateField() method increases the number of columns retrieved from the database when a query is performed, use it only when absolutely necessary. As more columns are retrieved, the number of joins and subqueries also increases, which can negatively impact performance.

- Use language exception handling features to write resilient code (eScript or Siebel VB).

  Both server script languages provide structured exception handling facilities. eScript has a try/catch/finally facility similar to C++; Java and Siebel VB have the On Error statement. Use these exception handling facilities for most nontrivial scripts to facilitate debugging and error logging or messaging.

  Nontrivial scripts are ones that could reasonably be expected to experience some type of error. Most often this includes—but is not limited to—scripts that query or write to the database, communicate with external systems, or interact with the host machine or the file system.

  What you should do when an error occurs depends on the design of the application, but generally you want to:

  - Clean up any object variables
  - Log off of any external systems or close any open sessions
  - Log the error to a file (or somewhere persistent) and notify the user

  Doing these operations will maximize the performance and stability of your application performance while minimizing debugging efforts.

  In eScript:

```
try {
   // Put code here that may throw an exception
}
catch ( objExc ) {
   // If an exception is thrown above, code will continue
   // in this block.  The exception object (objExc above)
   // will contain additional information about the error
   // that occurred:
  TheApplication().RaiseErrorText ( "Exception thrown: " + objExc
);
}
```

```
finally {
   // This block will always be called whether an exception is
   // thrown or not.  Put cleanup code here, e.g.:
   bcContact = null;
   bcAccount = null;
   boAccount = null;
}
```

In Siebel VB:

```
On Error Goto Cleanup
   ' Place code here that may experience an error
Cleanup:
   ' If an error occurs, we end up here.  We can use the
   ' Err and Error functions to get the error code or error
   ' text.  This is also where object variables should be
   ' cleaned up.
   Set bcContact = Nothing
   Set bcAccount = Nothing
   Set boAccount = Nothing
```

For more information on eScript error handling, see *Siebel Tools Online Help, MidMarket Edition*. For more information on Siebel Visual Basic error handling, see *Siebel Tools Online Help, MidMarket Edition*.

- Avoid nested query loops (eScript, Siebel VB, or Object Interfaces).

  Nested query loops are ones in which a parent business component is queried, iterated through, and then at each record a child business component is queried. Depending on the number of records returned by each query, the performance degradation from this technique can range from minimal to substantial.

  Use the nested approach only as a last resort. Never use it in a user-driven activity when the user must wait for the operation to complete. If this type of work must be done, it is most often because of some required aggregate function or bulk update. If such database-intensive processing is necessary, you should research the possibility of using a business service that is invoked from a Workflow process. If it is driven by user events, it can be triggered by a Runtime event or a Workflow Policy. This can allow the processing to take place asynchronously from the user operations.

  Also, be aware that nested query loops can also occur implicitly. When two or more business components are instantiated within the same business object and they are constrained by a link, advancing through the records on the parent business component forces requeries of the child business components. For example, in the following code, the Order Entry - Line Items business component (child) is refreshed every time NextRecord() is invoked on the Order Entry - Orders business component (parent):

```
var boOrder = TheApplication().GetBusObject( "Order Entry" );

var bcOrder = boOrder.GetBusComp( "Order Entry - Orders" );

var bcItem  = boOrder.GetBusComp( "Order Entry - Line Items" );


bcOrder.ClearToQuery();

bcOrder.ActivateField( "Order Number" );

bcOrder.SetSearchSpec( "Status", "Pending" );

bcOrder.ExecuteQuery( ForwardOnly );


if ( bcOrder.FirstRecord() ) {
```

```
   do {
      // ...
     // When the next line is invoked, the Order Entry - Line Items
      // BC is re-queried to enforce the link
      // "Order Entry - Orders/Order Entry - Line Items" in the
current BO
   } while ( bcOrder.NextRecord() );
}
```

This will not cause a problem if the parent record only returns one row. However, when searching by any attributes that do not uniquely identify the parent record, the performance degrades proportionally to the number of parent rows returned. If the child business component is also the parent of another business component, another query is issued for each advance of the parent business component record. The simplest alternative to the problems caused by the above is to move the instantiation of the Order Entry - Line Items beneath the loop.

- Use standard Siebel tracing functions (eScript or Siebel VB).

  When logging informational or debugging messages to a file, use standard Siebel tracing methods instead of the I/O facilities of the current script language. Your application comes with the Trace() application method, which can be placed throughout code to log messages and other useful information to a file. You can also use it to trace SQL or object allocation as well as developer-defined messages.

  The following example shows how to use it to trace small segments of code:

```
TheApplication().TraceOn( "d:\SEA7\Log.txt", "Allocation", "All"
);
// do something...
TheApplication().Trace( "My debug message" );
TheApplication().TraceOff();
```

For more information about the parameters for the TraceOn() method, see *Siebel Tools Online Help, MidMarket Edition*.

When there are different trace statements throughout application modules, put the TraceOn() method in the Application_Start() event and TraceOff() in Application_Close(). This causes all trace statements in between to be logged to the trace file. You can also use macros with TraceOn() to uniquely identify the output files based on the process and task identifiers used by the Siebel Server.

Using the standard file I/O features of the host language is often an appropriate solution, but you may have to manually write the logging functions. In addition, because multiple clients can run on one Object Manager, any custom functions for persistent logging must also include a strategy to manage concurrency.

- Always use Option Explicit (Siebel VB).

  Put the Option Explicit statement in the declarations section of every module that contains Siebel VB. When Option Explicit is used, the compiler in Siebel Tools enforces strict type checking, which requires explicit variable declaration. Therefore, if variables are not declared, the compiler generates an error message. Without the Option Explicit statement, misspelled variable names are instantiated on first use, which can require debugging.

  You can use the Option Explicit statement by adding it to the declarations section of any given module, as follows:

```
Option Explicit
```

- Follow a standard naming convention for all variable names (eScript or Siebel VB).

  To simplify maintenance, variable names must follow a standard and consistent naming convention. Nonexistent, inconsistent, or inadequate naming conventions may lead to confusion when enhancing, maintaining or troubleshooting code. Siebel scripting languages are loosely typed (eScript more so than Siebel VB), but you should indicate the types of variables represented in the variable name. Also, include the scope of the variable, because it is not readily apparent when reading code that it has been declared outside the scope of the current function. Here are some recommended naming conventions for eScript and Siebel VB variables.

  To indicate the scope and type of variables in their naming, use the following convention:

  ```
  [scope][type][var name]
  ```

  If the scope and type are one-letter abbreviations, use these conventions:

  ```
  g = global scope
  ```

  ```
  m = module scope (accessible only within the current module, e.g.
  Account business component)
  ```

  Variables with local scope do not require a prefix. The global scope applies only to Siebel VB, as eScript has no equivalent notion of application-global variables.

  Variable types should be descriptive without becoming overly granular. Here are some suggested conventions for type prefixes:

  ```
  i = integer
  ```

  ```
  s = string
  ```

  ```
  bc = business component
  ```

  ```
  bo = business object
  ```

  ```
  bs = business service
  ```

  ```
  ps = property set
  ```

```
o, obj = a non-Siebel object such as those returned by
COMCreateObject() or GetObject()
```

Here is an example when using eScript:

```
var sId;

var boOrder = TheApplication().GetBusObject( "Order Entry" );

var bcOrder = boOrder.GetBusComp( "Order Entry - Orders" );

var bcOrderLineItem = boOrder. GetBusComp( " Order Entry - Line
Items" );

var sOrderNum;

var iCount = 0;
```

Failure to consistently use a standard naming convention could lead to confusion when reading code. This could lessen the application's quality and increase the work required during the configuration process.

- Declare variables at the beginning of functions (eScript or Siebel VB).

  Declare variables at the beginning of functions to make it easier to read the code. It also becomes much easier to destroy variables if they are all declared in one place.

- Use the this and Me objects instead of TheApplication().ActiveXXX() methods (eScript or Siebel VB).

  Use the application methods ActiveBusObject() and ActiveBusComp() (browser script only) only for script written for user interface objects, such as applets, and not for business components or business services. Both methods will return references to the business object or business component that is the basis for the currently active user interface constructs, which causes errors or failure in nonuser interface contexts. In other words, the ActiveBusObject() method returns the business object on which the current view is based and the ActiveBusComp() methods returns the business component on which the currently active applet is based. Both methods require the presence of the user interface to return a value; otherwise, they will return a null reference.

  In most cases, use the this and Me objects instead of the active methods. Scripts written in any module should refer to these objects instead of obtaining references with the Active…() methods. Table 8 lists the available alternatives to using self-referencing objects.

Table 8. Alternative Syntax Using Self-Referencing Objects

Module	Target Object	Alternative Syntax (eScript or Siebel VB)
Applet	Current applet	`this` `Me`
	Current business object	`this.BusObject()` `Me.BusObject()`
	Current business component	`this.BusComp()` `Me.BusComp()`
Business Component	Current business component	`this` `Me`

As of Siebel 7, the ActiveBusComp() method can only be used from browser script and is not available in server script. In previous versions of Siebel eBusiness Applications, the ActiveBusComp() method could be invoked from any script, because there was no distinction between server and browser script.

When using the ActiveBusObject() method, it obtains a reference to the current view's business object instance. Business components retrieved from this instance with GetBusComp() similarly return references to the business components in the same context. These are the business components that are the basis for the currently active user interface objects.

In a nonuser interface context, like any of the object interfaces (for example, JDB, XML Web Interface, or COM interfaces) or within workflow processes, there is no active user interface, and these methods return null references. As a result, scripts written on business components or within business services will fail or generate error messages. This can confuse users and increase troubleshooting times.

- Be aware of the context in which objects are created (eScript, Siebel VB, or Object Interfaces).

  Use the correct context for business components and business objects based on the operations being done. The current context is the context in which a script is executing. For example, in a script on the Account business component's PreWriteRecord event, the following line of code will set the variable boAccount to the current instance of the Account business object:

  ```
  var boAccount = this.BusObject();

  var bcAction = boAccount.GetBusComp( "Action" );
  ```

  If this script executes because of a user event done in the user interface, then the business object returned is the one that the current view is based on. If the script is invoked from a nonuser interface context, it returns the business object instance that was created to do this operation. Therefore, operations on the current instance of an object (when dealing with user interface-based events) are reflected in the user interface. This includes SetFieldValue(), ExecuteQuery(), and NextRecord(). These are some of the methods that could cause undesirable activity in the user interface.

  To avoid this user interface activity, you can create a new business object instance separate from the one that the user interface is currently using. This allows more flexibility in the types of operation you can do without the possibility of updating the user interface. Do this as follows:

  ```
  var boAccount = TheApplication().GetBusObject( "Account" );

  var bcAction = boAccount.GetBusComp( "Action" );
  ```

This avoids modifying the user interface if changes are made, but this can cause problems because the timestamp of the last update on a record is used to manage concurrency. As a result, if a separate instance of a business component has a particular record updated and that record currently displays in the user interface, the user interface must be refreshed or the user will get an error message when manually updating this record.

It is recommended that you update the current instance of a business component when simple field updates are required for the current record, and that you use a separate instance when updating or iterating through multiple records. When using a separate instance, refresh the user interface if it is possible that the displayed records may have been updated.

If the user interface is not updated when the displayed records were modified through another instance of the business component, the user receives an error message when trying to update it. If a new context is used unnecessarily, it results in additional database activity.

- Put code in the most appropriate modules (eScript or Siebel VB).

  Write scripts on the most appropriate objects. You can write scripts for applets, business components, and business services.

  - Use applet scripts for specific functionality that is only invoked directly by users, such as pressing a button or responding to a similar user event. The benefit of using applet scripts is that they can be invoked from one place, so they provide a single place to put functionality that is not used elsewhere. In Siebel 7, applet browser scripts provide a powerful way to interact with the desktop machine, use browser capabilities directly (such as opening a new browser window), and do any other specific client-side logic. However, applet scripts cannot be reused or made generic.

    To centralize the script code, limit your use of applet script as much as possible. Before deciding to implement functionality using a script, first consider whether the script can be made generic. If it can be made generic, put it on a business component or a business service.

- Business component scripts are more general than applet scripts and should include any code that contains business logic related to the underlying business component. Scripts written as business component modules are executed without regard for the current user context, and can be used to reliably enforce typical business rules. Examples of these business rules are field validation, change propagation, or custom processing logic such as calculations or administrative functions.

  Business component scripts include code written in the predefined events (such as PreWriteRecord, PreDeleteRecord, or SetFieldValue) and custom methods you can invoke indirectly using the PreInvokeMethod event. For example, you could create a custom function called CloseAllChildSRs() for the Service Request business component that finds all child Service Requests for the current record and closes them. To do this, create the custom method in the Tools Script Editor, then update the BusComp_PreInvokeMethod() function for the Service Request business component as follows:

```
function BusComp_PreInvokeMethod (MethodName)
{
   switch (MethodName) {

   case "CloseAllChildSRs":

      CloseAllChildSRs();

      // Return CancelOperation, otherwise Siebel will
      // attempt to invoke the method against the BC's
      // C++ class, which will cause an error.
      return (CancelOperation);
   }
   return (ContinueOperation);
}
```

You can then invoke the custom function in one of two ways. First, it can be directly invoked by another script with the Invoke Method() business component method:

```
var boSR = TheApplication().GetBusObject( "Service Request" );
var bcSR = boSR.GetBusComp( "Service Request" );


bcSR.InvokeMethod ( "CloseAllChildSRs" )
```

It can also be invoked directly from a button on the user interface as shown in Figure 2. To do this, the applet must be based on the target business component, which in this example is the Service Request business component, and the Method Invoked property of the control must be set to the custom method name.

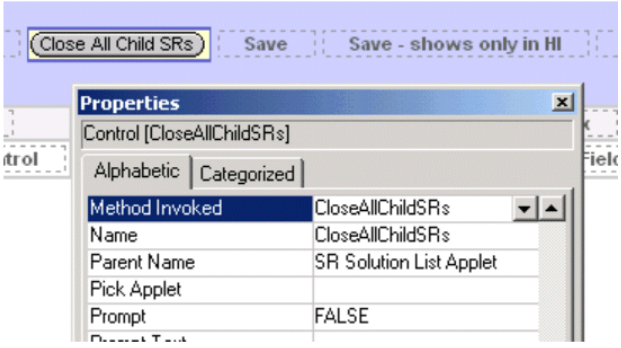

Figure 2. Invoking a Custom Function from a Button on the User Interface

Table 9 lists the standard uses and warnings for all available business component events.

Table 9. Business Component Events Standard Uses and Warnings

Event	Recommended Usage
PreSetFieldValue, SetFieldValue, PreNewRecord, NewRecord, PreAssociate, Associate, PreWriteRecord, WriteRecord, PreDeleteRecord, DeleteRecord	Typically, use the Pre… events to validate changes the user makes and cancel the operation if there is a problem. Only write code for these events that will determine if the event can proceed or not. Use data-driven read-only user properties to enforce validation whenever possible. (For more information, see *Siebel Tools Reference, MidMarket Edition*.) Use the post-processing events for logic that does not determine whether the event should complete. In the post-processing events (those that are not prefixed with Pre), limit code to processing that cannot be done through configuration. Investigate the use of Runtime events, Workflow, user properties, and calculated fields before writing script.
PreGetFieldValue	Modify field values before the value is retrieved by the GetFieldValue business component method. Do *not* overuse this event because it can cause severe performance degradation.
ChangeRecord	Use only for updating module variables or other short and simple scripts. Do not put code in this event that queries or writes to the database, contacts external systems, or interacts with the file system. This will result in severe performance degradation.
PreQuery, Query	Use the PreQuery event only for logic that cannot be achieved through search specifications.
PreInvokeMethod, InvokeMethod	Use the PreInvokeMethod to deal with custom methods. In some situations, you can use this method to capture standard Siebel methods and cancel them if necessary. Be aware that this is what the above events (such as PreSetFieldValue or PreWriteRecord) are for and code that depends on nonstandard methods, such as those used by specialized business components, may not properly upgrade. Use the InvokeMethod event to do any processing that does not determine whether the method invocation should continue. Avoid using InvokeMethod to handle methods for which there is a predefined event.

- Use business service scripts to group related functionality in generic modules. Use business service scripts to centralize functionality that will be used in multiple locations. For example, you can invoke business service scripts from applets, business components, Workflow processes, and other business services.

  The recommended way to add methods to a custom business service is to create the methods as you would any other script, and use the PreInvokeMethod event to act as a dispatcher that handles method invocations by calling the correct internal method. The following is a sample PreInvokeMethod script:

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
   switch (MethodName) {

   case "MyCustomMethod":

      MyCustomMethod( Inputs, Outputs );

      // Return CancelOperation, otherwise Siebel will
      // attempt to invoke the method against the BS's
      // C++ class, which will cause an error.
      return (CancelOperation);
   }
   return (ContinueOperation)
}
```

As with business component methods, PreInvokeMethod should return CancelOperation to prevent the method from being invoked against the C++ class of the business service. The two Inputs and Outputs property sets contain all input and output arguments. This is the only way arguments may be passed to business services.

Avoid placing all logic in the PreInvokeMethod event, as this makes the code less modular and more difficult to maintain and troubleshoot.

- If users need to inactivate a button when no records display, as a workaround, you can use the WebApplet_PreCanInvokeMethod event script to make the button visible. For example:
  - Add a button to an applet, and set the property Method = My Method
  - Modify WebApplet_PreCanInvokeMethod.

  Parameter CanInvoke must return TRUE for the button to be enabled. Otherwise, it will be disabled.

  The following sample code is used to enable the button if there are records in the applet. If no records are displayed, the button will be disabled.

```
function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke)
{
   if (MethodName == “My Method”)
   {
      // Set Id field to string
      var oBC = this.BusComp();
      var sId = oBC.GetFieldValue(“Id”);
      //Check value of field Id for current bus comp
      if (sId==””)
      {
      // if this is true there are no records in applet. Do not
      invoke the button return (CancelOperation);
```

```
oBC = null;
}
else
{
CanInvoke = “TRUE”;
return(CancelOperation);
}
}
else
return (ContinueOperation);
}
```

For more information about the WebApplet_PreCanInvokeMethod event, see *Siebel Tools Online Help, MidMarket Edition*.

Browser Script

The following section only applies to browser script. JavaScript is the only supported language for browser script.

- Avoid direct Document Object Model (DOM) manipulation of Siebel objects.

  The Siebel High Interactivity Client makes extensive use of the Document Object Model. Because browser script runs within this framework, it can access the DOM and all of the objects that constitute the Siebel High Interactivity Client. This access is not supported and could cause problems because the structure of these objects is not guaranteed to be static. Whenever you require access to Siebel user interface constructs, use the Siebel Object Interface methods.

- Do not return large result sets from server business services to browser script.

  Browser script can invoke methods on business services that are either browser scripts or server scripts. Because browser script supports a subset of Object Interface methods available to server script, it is often convenient to write business services with server scripts that are invoked by browser scripts. Using this technique, browser scripts can effectively invoke server-side scripts. While this is a good solution, these types of server scripts should return simple values or a single record to the calling browser script, not sets of records returned from a query.

  For example, the business component method ExecuteQuery() cannot be directly invoked using browser script. To get around this restriction, instead of writing a server business service that takes query criteria and a business component name as arguments and then returns the results, return success or failure or the number of rows found.

  Returning entire result sets to browser script can significantly decrease performance. Server script must first convert data to a property set, where it is encoded as a string by the Siebel Web Engine (SWE). Then, it is sent to the SWSE using SISNAPI, and returned to the client through HTTP, decoded back to a property set by JavaScript, and used by the browser script. For large result sets, this negatively impacts performance.

  The user interface can efficiently display large result sets (for example, the All Accounts view, where there may be thousands of accounts in the view) because it retrieves only the number of rows from the server that can be displayed on the current applet.

  Often, if browser scripts need to invoke server-side functionality, you can simply write the script in server script on the corresponding event instead of browser script.

Configuring the User Interface 7

This section gives guidelines for configuring the user interface.

User Interface Guidelines

These guidelines apply to the Siebel eBusiness Application user interface for both Standard Interactivity and High Interactivity. However, be aware that there are a few differences between these two clients. For example, because the High Interactivity client is considered the standard application, it provides almost all the functionality that was available in previous versions of Siebel eBusiness Applications. However, the Standard Interactivity client, with its reduced level of interactivity, is considered to be more of a customer application. For more information on the differences between the two types of clients, see *Siebel Tools Reference, MidMarket Edition*.

Configuring the user interface represents a significant part of any Siebel eBusiness Application implementation. Because the user interface is made up of many different parts, this section provides guidelines for the following:

- User Interface
- Template Files
- Performance
- Screen Design
- Personalization
- Deployment

Standard Siebel eBusiness applications are deployed using Siebel Smart Web Clients that have their own separate set of user interface definitions. These definitions include applets, views, screens and application object definitions.

User Interface Object Definition

A standard Siebel application has a set of user interface definitions that are unique to each application within the repository. These user interface definitions are controls, applet Web template, view Web template and Web page. The applet Web template object definition is a child of the applet and the view Web template object definition is a child of the view definition. After you have defined the standard applets and views, you can create or modify the corresponding Web template definitions.

Some reasons to create a complete set of user interface object definitions include the following:

- Performance is more of an issue because the number of fields in an applet has been reduced. To improve the database engine and the Siebel Web Engine (SWE) performance, retrieve required fields only.
- The Siebel Smart Web Client uses the properties defined in the underlying view and applet definitions. For example, the visibility settings in the view are used by their corresponding view in the Siebel Smart Web Client. In addition, any search specification or control-specific properties apply to an applet used by the Siebel Smart Web Client.
- Because the user interface has changed significantly, the economies of reuse do not always apply.
- The corresponding Web template components are specific to the application. For example, the Account applets used in the eChannel application are different from the Web Phone Account applets.
- Siebel Industry Applications (SIA) introduced the concept of conditional control mappings. This means that you can use the same applet template in different applications (for example, eClinical and eLifeSciences), and display different controls on the same Web template. An eClinical application user sees different fields on the Account List applet than someone using the eLifeSciences application, even though both applications are using the same SRF. This is accomplished through the Expression property of the Applet Web Template Item, and it allows more than one control to be mapped to the same template ID.

Implementing the User Interface

After defining the screen flow and required set of user interface components, complete these steps in this order:

1. Configure the applets, views, screen, and application objects.
2. Create and modify the required controls, applet Web templates, view Web templates, and Web pages object definitions.
3. Compile and test the application.

Follow these guidelines to implement the user interface:

- Follow the object naming conventions.
- For minor changes to applets or views, reuse the existing applet or view definitions.
- For major changes to applets or views, copy the existing applet or view definition, rename the applet or view, and make the appropriate modifications.
- When cloning an applet, be sure to set the applet's Upgrade Ancestor property. Setting this property will make sure that any future functionality enhancements to the base applet (such as new Web templates, new Web template items, or new specialized applet class functionality) are incorporated into the cloned applet after you upgrade your application.
- You can create custom HTML controls. These controls are defined in the SWF file that is contained in the application CFG file. Be aware that creating custom HTML controls requires changes in the list of values used by the Siebel repository. For more information, see *Siebel Tools Reference, MidMarket Edition*.
- Set a specific sequence for each View Web Template Item in a view. This sequence defines the order that the view applets will appear in the user interface. Also, set the view Applet Mode property on a View Web Template Item, in case the associated applet contains more than one Applet Web Template and simultaneously supports multiple modes. For example, a High Interactivity View Web Template Item could be set to the Edit List applet mode for the Account List applet, while a Standard Interactivity view using the same applet could specify the Base mode as the default mode for the same applet.

- Controls that have a MethodInvoke property to go to a view or execute a SmartScript often contain user properties (for the List Column or Control definition). For examples, check in the repository.

- In the Siebel Smart Web Client, SmartScript pages can be broken into several pages. This is because any question containing script must be treated as the end of a page so that the Object Manager can do any processing associated with the script before the next question is answered. For example, if a page contains three questions (q1, q2, and q3) and two of these questions have question scripts (q2 and q3), then two pages are created—one with q1 and q2 and the other with just q3. To avoid this, consider consolidating the script into the last question on the page using the Question_PreBranch or Question_Leave events. In the preceding example, question q3 would contain all of the necessary script.

Template Files

Template files are files that have both Siebel tags and HTML tags. A standard set of template files comes with every Siebel eBusiness application. They are installed during the Siebel Tools and Siebel Server installation in the WEBTEMPL directory.

Follow these guidelines when using template files:

- Create a new template file if you are making changes that will result in destroying or altering the standard functionality of your application.
- Always try to reuse any new or modified template files. Too many unique template files often indicates inadequate screen design.
- Do not use a new template if a standard template will satisfy your requirements.
- Create a new container page only if changing a container page template file will break or alter the functionality of your application. Otherwise, modify the file.
- If changing a view template file affects all views, then modify the existing template. Otherwise, create a new view template file to meet your requirements.
- If changing an applet template files affects all applets, then modify the existing template. Otherwise, create a new applet template file to meet your requirements.
- Do not use frames on the templates. If you use frames, you will break the browser functionality for bookmarking and paging backwards and forwards because of the difficulty in maintaining sessions.

Before Modifying Templates Files

Although modifying templates for employee applications such as Siebel Sales and Siebel Call Center is typically not necessary, it is common to need to modify Web templates for customer and partner applications. For example, you may want to add a corporate logo to every page seen by your customers or partners.

When modifying Web templates, consider the following:

- For Siebel employee applications the standard templates are tightly integrated with the High Interactivity Framework. Modifying or creating new templates may cause unexpected behavior which is difficult to troubleshoot.
- Siebel Web templates files, like cascading style sheets, are not automatically upgraded. Consequently, if you make modifications to Web templates, the modifications will need to be reapplied manually after upgrading to a new release.

Before you modify Siebel Web templates, consider how you might configure objects definitions in Siebel Tools or use other means to meet your user interface requirements. For example:

- Reposition fields and other controls, such as buttons, by dragging and dropping them to different locations in a template using the Web Template Layout Editor.

  See *Siebel Tools Reference, MidMarket Edition* for more information.

- Use the functionality provided by the More/Less feature to add additional fields to an applet but only display them when a user clicks the More button in the upper right corner of an applet.

  See *Siebel Tools Reference, MidMarket Edition* for more information.

- Use Siebel Conditional tags to target different layouts for different browsers. Conditional tags are Siebel tags that process *if* or *switch/case* logic. They allow you to target different layouts depending on which browser type (for example, Internet Explorer 5.5 or Navigator 6) is making the request. A single template can handle layouts for multiple browser types.

  See *Siebel Tools Reference, MidMarket Edition* for more information.

- Use Siebel Web Format file (.swf file) to define a new HTML Type. For example, you could create a new HTML Type called LargeButton by copying the standard MiniButton and then changing the size and image used for the button. After you add LargeButton to the List of Values, it will be available to select as the HTML Type property of a control.

  See *Siebel Tools Reference, MidMarket Edition* for more information.

If you must modify or create Web templates, try to keep the number to a minimum; the fewer, the better. Here are some general guidelines to help achieve this goal. The following list is ordered by the amount of effort that will be required to upgrade to a future release—the first item will require the least effort and the last item will require the most effort.

- **Modify the container page**. There is only one container page in an application, so it is acceptable to make changes in this template. This is a relatively easy way to make changes that apply throughout the application. For example, you might add a new logo or other navigational links to the template.

- **Modify view templates**. There are only a few view templates in an application. It is generally acceptable to add navigational links in view templates. Changes in a view template affect all View objects that reference the template.

- **Modify applet templates**. There are a number of applet templates in an application. It is best to minimize changes to applet templates. In most cases, the layout placeholders in the standard applet template are flexible enough to meet your needs.

- **Hardcode applet templates**. This is writing your own HTML directly into a template, such as hardcoding the width attribute of a < table > tag. Doing this may cause unexpected behavior. For example, it might change the template so that it is no longer reusable, which would lead to a proliferation of applet templates that would have a severe impact during upgrades.

- **Create new templates**. New templates can be useful when they are used for very specific purposes. However, you should be careful to avoid creating numerous instance-specific templates because of the consequent effort required to manually upgrade each one. If a new template is necessary, it should be a copy of a standard template, and the copied template should have the same set of Siebel tag IDs as the original. This will allow easy replacement because existing association information between tag IDs and UI objects in the Repository will continue to work.
- **Embedding JavaScript in templates**. This should be avoided. See *Siebel Tools Reference, MidMarket Edition*.

For a complete description of each standard template, see *Siebel Tools Reference, MidMarket Edition*.

About Embedding JavaScript in Web Templates

It is not necessary to embed client-side JavaScript in Web templates. Instead, you can write client-side JavaScript in Siebel Tools using Browser Script. Browser Script allows you to interact with browser objects, such as form fields, and interact with Siebel objects, such as business components and business services. The advantage of using Browser Script is that, because it is developed in Siebel Tools, changes are stored in the Siebel Repository. And since Browser Script is not hardcoded into a template, applets with Browser Scripts can share the same template.

For more information about Browser Scripting, see *Siebel Tools Reference, MidMarket Edition* and *Siebel Tools Online Help, MidMarket Edition*.

The general recommendation is to avoid embedding any client-side JavaScript in Web templates. Doing so not only limits the reusability of Web templates, but may also adversely affect Siebel employee applications that run in the High Interactivity Framework, which uses client-side JavaScript extensively.

Cascading Style Sheets

Modifying standard .css files and classes allows you to change the look and feel of the user interface. This is more efficient than creating new ones because it allows you to apply changes to all Web templates that reference a particular .css class. Creating new .css files and classes would require you to also modify each Web template that needed to reference the new class. Keep in mind that changes to standard .css files are not automatically upgraded. For this reason it is good practice to keep the number of modified templates and .css files to as few as possible. This will help minimize the effort required to upgrade to future releases.

For more information on cascading style sheets, see *Siebel Tools Reference, MidMarket Edition*.

Performance

Consider the following suggestions to reduce the size of the generated HTML page. To make template files smaller:

- Remove comments.
- Remove references to spacer.gif.
- Remove double quotes on HTML tags attributes.
- Remove empty lines.
- Remove default values (border = 0, alt = ”“ …).
- Remove tabs and blanks at the beginning of the lines.
- Shorten folder and file names for the images (*.gif).
- Remove other default values (valign = middle).
- Optimize HTML (for example, move some attributes of the cell to the table tag, remove some properties declared on the table and on the cell).
- Remove comments and double quotes on CCHtmlType.swf.
- Remove double quotes on Siebel Tags attributes.
- Remove bgcolor attribute on mini-buttons.
- Remove logos at the top of the page.
- Remove vertical separator in the list applets.
- Simplify the style sheet used (main.css) wherever possible by adding or modifying the classes.
- Avoid using animated GIFs.

Do the following to simplify HTML configuration:

- Reevaluate responsibilities to limit items in the screen tab bar.
- Include required fields only in a list applet and limit the number of columns as much as possible.

- Reduce the number of rows in a list applet using the HTML Number of Rows property on the applet object.
- Eliminate or limit the use of custom JavaScript within the templates.
- Because they use the HTML option tag, limit forms using multiple static picklists—they increase the size of the page significantly.
- Evaluate complex conditions in the personalization module that could degrade performance for affected applets, and consider using new conditional tags if necessary. For more information, see *Siebel Tools Reference, MidMarket Edition*.

Follow these guidelines when using non-Siebel technologies:

- Because they can add to the download time, do not overuse Shockwave or Flash-type technology. Focus on the site's usability, not its appearance. If using these technologies sacrifices performance, resulting in poor user acceptance, reconsider using it.
- Some clients use compression techniques between the Web server and the client Web browser. This requires you to install third-party software on the Web server and Client machines. Clients should thoroughly test these compression technologies before deploying them with their Siebel eBusiness application.

Screen Design

Screen design impacts the development or configuration stage of an implementation. By keeping designs simple and consistent, you can configure your application effectively and efficiently.

Consider these performance recommendations:

- Limit the number of applets in a view and the number of fields that appear on List applets. Your overall objective is to let customers see as much relevant data in a view as possible without scrolling.
- As users of the Standard Interactivity client may not receive any training, make the interface as intuitive and simple as possible. Include as much Help text in the HTML page as you can to make user interaction easy.
- Limit the number of columns and return only the information that is absolutely necessary.
- Consider using drilldowns more extensively to access detailed information. This helps limit the amount of information.
- For forms using SmartScript, try to limit the number of questions asked to between five and seven questions per page. Limit the number of pages as much as possible.
- Avoid nonstandard link colors.
- Keep information consistent across the applet templates that are used in the base, edit, and editlist modes.
- Limit the use of Personalization rules to the user's home page.
- Avoid using animated GIFs as much as possible.

Personalization

You can use personalization to increase user satisfaction. However, excessive personalization can have a negative impact on performance. The following guidelines apply to personalization when configuring your application:

- Test personalization rules thoroughly to make sure that they do not override your application's desired behavior.
- Do not use personalization to replace standard visibility and the use of search specifications on business components or applets.
- Avoid using personalization to trigger general validation scripts or to invoke custom methods that implement application logic. This functionality should be confined to the Business Objects layer through standard configuration techniques or scripting.
- Carefully manage the profile attributes set in a given session. Maintain personalization rules using the Personalization Administration and Runtime Events Administration screens. Setting attribute values through scripting in the repository could result in conflicts and unexpected behavior.
- Using SetProfileAttr and GetProfileAttr can be effective for reasons other than storing user profile information. For example, calculated fields can use expressions containing GetProfileAttr. Search specifications can also use expressions with GetProfileAttr. Profile attributes in this case are global variables stored in the application, so they need to be documented and appropriately managed in order to avoid conflicts and errors.

Deployment Issues

Review the following issues and considerations when deploying your Siebel eBusiness application.

- The ratio of MaxTasks and MaxMTServers should be less than 1:10. Generally, do not have more than five tasks for each Object Manager.

Some of the security issues you will need to consider are:

- Is single sign-on required on the customer Web site? This is now a supported technology.
- How is security being implemented? Is secure-socket layer (SSL) being implemented?
- How are you managing new user registration in the Standard Interactivity client? This will impact the Login template and user administration.
- Are you using a Lightweight Directory Access Protocol (LDAP)? Is a custom security adapter required?
- Are there requirements to integrate with a communication server?
- What are the architectural concerns, such as firewall implementation or protocol?
- Are there any legacy integration issues—for example, is eBusiness Application Integration (eAI) necessary?

Creating User Interface Objects

This section provides guidelines for creating screens, views, and applets.

Screens

A screen object is used to group or modify a collection of views.

Creating and Modifying Screens

Typically, you create a new screen when you create a new business object. You modify an existing screen object when you remove or add a screen tab, change menu text (for the Site Map), or define new views for that screen. Screens appear as first-level choices on the Site Map.

Views associated with screens appear in one of two places. Views with visibility properties (specifically the Visibility applet) appear in the Site Map as views accessible under the screen. They also appear in the Show drop–down list in the screen's default view. Screens are also accessed through screen tabs. Each screen tab represents a screen. The views that are part of that screen are available in the Show drop–down list (those with visibility properties) or are represented as view tabs under the screen's default view.

The order of the tabs and their listing on the Site Map are controlled by the sequence property of the records in the page tab and screen menu item child object of the application object.

Unused Screens

If you will not use a standard screen in your implementation, use the Responsibility Administration Screen to disassociate all views on the redundant screen from those responsibilities your organization uses. This approach reduces the amount of configuration necessary for you to maintain and upgrade. 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—it will not be compiled in the SRF.

Views

Typically, a view defines a visual representation of a business object's data and is composed of a collection of applets. There can be many views of information for a business object.

Creating and Modifying Views

During the initial stages of an implementation, you define the required views. These views are usually in one of these categories:

- Views that are standard and do not need to be modified.
- Views that closely align with an existing view and require the same applet layout, but require other configurations such changing a title, inactivating or adding controls or list columns, and changing display labels. For this category, it is recommended that you configure the existing view object. Most configurations in this category are actually done on the applets instead of the views.
- Views that closely align with an existing view but require a moderately different applet layout, possibly displaying applets based on new business components or adding toggles. For this category, it is also recommended that you configure the existing view object. The design may also require configuration of the existing view applets.
- Views that consolidate two already existing views. For this category, it is recommended that you configure one of the existing views by modifying the view object, and remove visibility to the redundant view using the Responsibility Administration screen.
- Views that do not have an obvious preexisting equivalent view. These are views that expose new functionality specifically configured for your implementation, exposing new business objects, or business components. For this category, it is recommended that you create a new view object and avoid extensively modifying an existing definition that will not be used in your implementation. The resulting configurations will be much cleaner and easier to maintain and upgrade (both manually and automatically).

A view can be associated with more than one screen, but this configuration 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.

Guidelines for Views

The following sections give guidelines for which views not to configure, naming views, handling unused views, and choosing a view layout.

Server Views

CAUTION: Do not modify Server Administration views. Information in these views is read from the siebens.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 siebens.dat file. However, it is not possible to configure the product to store such information in siebens.dat. Therefore, configuration of server views is not recommended or supported.

Threads

Views associated with more than one screen in a given application will cause incorrect behavior in Siebel applications. When the thread is saved in the session file, the name of the view is saved without the name of the associated screen. When the end user chooses a thread that navigates to a duplicated view, the Siebel applications will always navigate to one screen only—even if the thread was created in the other screen. Furthermore, if the duplicate view is defined as the default view on both tabs, the end user will see an anomaly in the user interface. Siebel applications will select one tab as the active tab when either of the tabs is selected. The duplicate tab will never appear to be active.

NOTE: The Thread Applet property must be correct, especially if a custom applet is placed in Sector 0. If this property is not set with the correct applet name, planned CTI screen pops for transfer calls will not work.

Unused Views

If you are not using a standard view object, use the Responsibility List Administration screen to disassociate the redundant view from any responsibilities your organization uses. This offers an easy upgrade path if you decide to expose the view later. At that time, no configuration or software upgrade is required; you need only to reassign the view to the relevant responsibility. You can also inactivate the view within Siebel Tools.

View Titles

There are three different titles displayed for a view, as follows:

- **Title bar of the Siebel application window.** The title appears in the title bar, prefixed by the application name and a hyphen, as in Siebel Sales - Account List View. This is specified in the Title property of the view.
- **View bar tab.** In the View bar in the appropriate screen, the tab that navigates to this view. This is specified in the Viewbar Text property of the corresponding screen view object definition.
- **Screens menu sub-option.** In the Screens menu, as a sub-option of the appropriate screen, the menu option that navigates to this view. This is specified in the Menu Text property of the corresponding screen view object definition.

Keep these three title definitions consistent for one view. If at all possible, the text should be identical in all three.

If a view specifies a visibility mode, as indicated by a non-blank Visibility Applet Type property, the title (in all three locations) needs to identify the visibility mode, as indicated in Table 10.

Table 10. View Titles by Visibility Mode

Visibility Mode	Title Format	Example
Sales team visibility	My *buscomp*(s)	My Contacts
Personal visibility	My Personal *buscomp*(s)	My Personal Contacts
Manager visibility	My Team's *buscomp*(s)	My Team's Opportunities
All visibility	All *buscomp*(s)	All Accounts

For more information on visibility modes, refer to Chapter 9, "Visibility," and *Siebel Tools Reference, MidMarket Edition*.

Applets

An applet is composed of controls and occupies part of a view. You can configure an applet to allow data entry, provide a table of data rows, and display business graphics or a navigation tree. It provides viewing, entry, modification, and navigation capabilities for data in one business component. An applet is always associated with one and only one business component.

Creating and Modifying Applets

During the initial stages of an implementation, you define the required views. The applets displayed on these views usually fall into one of these categories:

- Applets that are standard and do not require any modification.
- Applets that closely align with an existing list applet, but require minor configuration such as a title change, inactivating or adding controls or list columns, and changing display labels. For this category, it is recommended that you configure the existing applet object and its child objects.
- Applets that represent an already existing relationship (for example, opportunity contacts), but require extensive modification of the existing applet to produce a new applet layout. Modifications are considered extensive if the new applet requires a large combination of different configurations, such as resequencing of existing controls or list columns, inactivating and adding new controls and list columns. For this category, it is recommended that you create a new applet object (by copying an applet that closely resembles the applet you want to create), and then modifying the copied applet. The resulting configuration will be much cleaner and easier to maintain and upgrade (both manually and automatically).
- You may need an applet with different drilldowns in different views.
- Applets that do not have an equivalent existing applet. These tend to be applets that expose a new business component. Always create a new applet for this category of list applet.

NOTE: Remember to set the Upgrade Ancestry property on all custom applets that are cloned from another applet.

Modify, rather than copy, an applet, unless you are making extensive modifications to the applet. This avoids having to change all references of that applet to the new copy.

The following are examples of situations in which you might need to copy an applet:

- When you must extensively modify an existing applet.
- When you need a Read Only copy of an existing applet.

NOTE: Do not change the Class property of preconfigured applets.

The objective of your design and configuration projects should be to produce a consistent and intuitive user interface. Wherever possible, applets displaying the same business component should be consistent across different screens and views. For example, the contact list displayed for an opportunity should be consistent with the contact list displayed for an account. Whenever possible, reuse applet definitions between different views and screens. Obvious exceptions include redundant controls or list columns and controls, or list columns that are relevant only to the current parent business component. For example, you would display the contact's account information when displaying opportunity contacts, but not when displaying account contacts. This recommendation does not necessarily apply when comparing list with form or entry applets. When the screen space for a view is limited, it may be practical to include fields at the end of a list applet that are not displayed on the associated entry applet.

Another approach to increasing the number of fields available in a form applet is to use applet toggling. You can define two applets based on the same business component to show information from the same record (but the field controls are distributed over multiple applets) and the user can toggle between the two.

Unused Applets

It is recommended that you do not modify any applet you are not using in your implementation. You could also mark the applet as inactive.

Online Help

If you want to provide context-sensitive help in your application, you need to define Help IDs for screens and views. If you do not want context-sensitive help and you do not define a Help ID, the default start page for help will appear. For more information on Help IDs, see *Online Help Development Guide, MidMarket Edition*.

Applet Titles

The applet title is the value in the Title property. It determines what displays in the tab at the upper left of an applet in a view, or in the title bar of a pop-up applet. 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 11 offers standard conventions for the titles of certain applet types.

Table 11. 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

Managing Repositories 8

This section describes the process of managing repositories.

Establishing a Development Environment

You need to establish a development environment that includes:

- A Siebel Web server
- A Siebel database server
- A Siebel application server

NOTE: The development environment *must* be completely separate from the production environment. After migrating to a production environment, do not perform any development work.

You must also have a separate testing environment. You can migrate your configuration to your testing environment for system testing before installing in a production environment. Like the development environment, your testing environment will include:

- A Siebel Web server
- A Siebel database server
- A Siebel application server

The development database stores the working copy of all repositories being configured by your developers. Configuration work should only take place on the development database. After you have finished configuring a repository, use the Siebel REPIMEXP utility (as part of the Database Server Configuration Utility) to transfer that repository to the test environment and production environment.

Naming Repositories

You must establish and maintain a naming convention for all your repositories in their respective environments. Several things depend on repository names—for example, both the Siebel eBusiness Applications client and application server programs point to a specific repository by name. Also, the procedures for upgrading to new versions of Siebel eBusiness Applications depend on repository names.

A consistent naming convention promotes successful configuration and testing while it minimizes the work required to migrate new repositories or perform upgrades. Follow these guidelines when determining the naming conventions for your repositories:

- Decide on a repository name for your production environment. By default, it is *Siebel Repository*. Change this only if you have a compelling reason, because much of the Siebel documentation and the default configuration of the Siebel eBusiness Applications assume this name is being used.
- Choose the same name for the active repository in your test environment and for the current working repository in your production environment. The default name is *Siebel Repository*. Using the same name simplifies the process of migrating repositories from development to test and from test to production. It also eliminates the need to change your client or application server configurations when you do so.
- Use descriptive names for the other repositories in your development environment. Typically, your development environment has a number of repositories in addition to the current repository that is being configured. These may include the initial repository loaded with your Siebel application, other repository versions used in Siebel application upgrades, and repositories from previous versions of your custom configuration. Give these repositories unique and fully descriptive names — for example, *Siebel v7.0 Original* for the initial repository shipped with Siebel eBusiness Applications version 7.0. For more information, see Chapter 5, “Performance Guidelines.”

Backing Up Repositories

Regularly create backups of the development repositories to safeguard your configuration work. A daily backup takes only a few minutes and helps prevent lost work. You can either back up the entire development database, using the utilities provided by your RDBMS vendor, or use the Siebel utility REPIMEXP.EXE to export the contents of a specific repository to a flat file. For more information, see *Siebel Tools Reference, MidMarket Edition*.

You can also use REPIMEXP.EXE to integrate Siebel repositories into your source code control environment. After you finish configuring a specific version of your application, export the repository to a flat file and keep this file in an archive or source code control system. Include other application components in the archive, such as client configuration files (CFG), Siebel repository files (SRF), batch scripts, and Enterprise Integration Manager configuration files (IFB).

Exporting and Importing Objects

The object definition export and import features in Siebel Tools allow you to copy object definitions from one repository to another repository. These features let members of development teams share pieces of the configurations they develop. Users can export object definitions from the source repository to an archive file, which has the SIF filename extension. This archive file can then be imported into another repository.

In addition, you can check the differences between objects being imported and those already in existence by using the Import Wizard in Siebel Tools. You can also override the importation of specific attributes with the Import Wizard. For more information, see *Siebel Tools Reference, MidMarket Edition*.

Controlling Sources

You can use the repository check-in and checkout mechanism in Siebel Tools to interact with a third-party source code control system, such as Microsoft SourceSafe or PVCS. This allows you to create an archive file of the project, which is checked into the source control system each time a project is checked into the server repository.

Delivering Patches

You can use the Patch feature to deliver patches to your organization for upgrading your repositories to a new release. You can also use this feature to distribute internal upgrades.

Visibility 9

This section defines visibility and details its various aspects.

Visibility Overview

Visibility is a very important concept in delivering applications that meet the needs of an enterprise. Generally, visibility is the terminology used for access control in Siebel eBusiness applications. This section discusses the benefits of using visibility, what visibility is, and how to configure visibility for different types of users, such as remote users. Additional information on visibility can be found in *Siebel Tools Reference, MidMarket Edition*.

Correctly implementing visibility in your application provides the following benefits:

- Users see only necessary views.
- Users see only necessary data.
- Visibility provides different types of ownership (for example, organizational, team-based, access group or personal).
- You have more flexibility in segregating and partitioning data.
- There are smaller mobile client database extracts.
- Mobile users get shortened extraction and synchronization time.

Access Control

Visibility is controlled through three major mechanisms:

- Login, which determines the user's identity
- Responsibility, which determines the views available to the user
- Position, which controls the data available to the user

You can use access control to provide additional control. It is implemented through five different mechanisms:

- Personal Access Control

  This mechanism allows you to associate individual data with the user's database record. Examples are My Service Requests or My Activities.

- Position-Based Access Control

  This mechanism allows you to associate individual data to a position that is assigned to the user. A single position or multiple positions (such as a Sales Team) can be associated to the record. Other positions can access the data if their position is tied to the position through a hierarchy, such as a manager. It is usually better to work with positions instead of individuals, because reassigning a position is easier than reassigning an individual. Examples are My Accounts or My Team's Accounts.

- Organization-Based Access Control

  This mechanism lets you further restrict access to a record by associating the record to an organization and then allowing access to only those active positions who belong to that organization. Examples are All Accounts or All Contacts. These views retrieve all the data for the organization of a user.

- All Access Control

  This mechanism provides access to all records with a valid owner. The owner can be a person, a position, a valid primary sales team position, or an organization. All users with a view in their responsibilities that applies All access control see the same data in the view. An example is All Accounts Across Organizations.

- Access Group Access Control

  This mechanism provides access to records where the user is associated with an access group if, during the current session, the user is associated with a position, organization, division, account, household, or user list that is a member of the access group. Access groups control master data only.

View Access Visibility

Responsibilities control view access, which determines the views available to a user. You use these responsibilities, which are accessed in an administration view, to assign one or more responsibility records to a user in a user's employee record. Views are also assigned to each responsibility in this view. The result of these two administration activities is that when a user logs on, that user's responsibilities and list of available views is determined. Only the views to which a user has access are made available to the user; all others remain hidden.

You define a responsibility for a class of users who require access to the same set of application features—for example, users who need to see the same data.

For more information on administering views and responsibilities, see *Applications Administration Guide, MidMarket Edition*.

NOTE: A user can have multiple responsibilities. When accessing an application, the user's responsibility is a combination of all defined responsibilities. Therefore, if View A is defined in Responsibility 1 and View B in responsibility 2, the user has access to both Views A and B.

Record Access Visibility

Record access visibility, or client-side visibility, is a mechanism that restricts the data records that appear when a user navigates to a particular view. You use Siebel Tools to implement this type of visibility by setting properties on the following object types:

- Business Components
- Views
- Links
- Drilldowns

From a technical standpoint, the interpretation of these object visibility properties determines part of the WHERE clause of the database query that is used to select the data.

Object Ownership Models

Record ownership of Siebel entities can exist according to four different models:

- **Personal.** The record has a foreign key to the table that stores user information (for example, S_CONTACT) identifying the user as the owner of the record or the individual assigned to the record.
- **Single Position.** The record has a foreign key to the Position Table (for example, S_POSTN) identifying the owner's position.
- **Team Access.** The record is associated to a team of positions through an intersection table (for example, S_POSTN_CON). One position is considered the primary.
- **Organization.** The record belongs to an organization that is associated to your position.

Therefore, the records a user can see is determined by answering the following questions:

- What is the organizational visibility of the underlying business component?
- Is it a team-based business component?
- Can the business component be owned?
- How is view, which displays the data, configured?

Configuring Visibility at the View Level

There are eight visibility types you can configure at the view level:

- **Personal.** Provides access to records owned by the user. Position or Owner Id is that of the current user.
- **Sales Rep.** Provides access to records for which the user is part of the team. This may be a sales team, contact access list, account team, or other type of team. The term *Sales Rep* may be deceiving because it may not be a sales-related team.
- **Manager.** Provides access to records for which the user or a subordinate is the primary. Primary position or Owner Id is that of the manager or subordinate.
- **Organization.** Provides access to records within the user's organization—for example, the organization of the record matches that of the position of the user. Records must have a valid primary position or they are not displayed.
- **Sub-Organization.** Provides access to data in two situations. In the first situation, if the business component on which the view is based uses single organization access control, the user sees data associated directly with the user's active organization or with a descendent organization. In the second situation, if the business component on which the view is based uses multiple organization access control, then the user sees data for which the user's active organization or a descendent organization is the primary organization.
- **All.** Provides access to all data except for records that do not have a valid owner.

- **Group.** Provides access to categories of master data that are associated with any of the access groups with which the user is associated, such as household, user list, or organization. In a view where you navigate using a tree applet, the user sees accessible first-level subcategories (child categories) in the current category. In a view that provides a list of master data records, the user sees all the records in the current, already accessed, category. To use this visibility applet type, the business component must have a view mode with an Owner Type of Group.
- **Catalog.** Provides access to a flat (uncategorized) list of all the data in all the categories across catalogs to which all of the user's access groups have access. Typically, this visibility type is used in product picklists and other lists of products. To use this visibility applet type, the business component must have a view mode with an Owner Type of Catalog category.
- **Admin Mode.** This is a separate visibility mode to provide a way to see all records, including ones without a valid owner or primary sales team member. This allows the administrator to fix records that would not otherwise be visible to anyone.

NOTE: Because you use the Admin Mode Flag property to set the Admin Mode for a view, it is not really a view visibility setting.

Naming Views

Typically, you should conform to a standard naming convention when naming views. The following examples are provided for each view visibility type. The underline identifies examples of the conventions you should use for each type of view visibility when naming views.

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*

View Visibility Type	Naming Convention
All	*All* Contacts *Across Organizations*
Group	*User Catalog* List View
Catalog	Products *across Catalogs*
Admin Mode	Contacts *Administration*

Configuring Visibility Using Siebel Tools

The following sections give more details on configuring team-based visibility settings.

This setting restricts access to members of a team for team-based business components. Examples of this setting include Opportunity, Contact and Account. You configure team-based visibility by setting the Owner Type, Visibility Field, Visibility Emp MVField, and the Visibility MVLink properties on the Business Component View Mode, which is a child object type of the business component. If you do not set the Owner Type, Visibility Emp MVField and the Visibility MVLink properties, these records will be visible to everyone.

The view visibility properties should be set to work with these Business Component View Mode properties as follows:

- My View, which displays all records for which a user is on the team, should have the following properties:
  - Set the Visibility Applet property to identify the first applet defined in the Base definition of the View Web Template Item for the view.
  - Set the Visibility Applet Type property to `Sales Rep` (this is the default value if none is set).
- My Teams View, which displays all records for which the manager or subordinates are the primary or owner, should have the following properties:
  - Set the Visibility Applet property to identify the primary applet in the view.
  - Set the Visibility Applet Type property to `Manager`.

- All View, which displays all of the records within the organization (with a valid primary set), should have the following properties:
  - Set the Visibility Applet property to identify the Sector 0 applet in the view.
  - Set the Visibility Applet Type property to `Organization`.
- All View Across Organizations, which displays all records across all organizations (with a valid primary set), should have the following properties:
  - Set the Visibility Applet property to identify the Sector 0 applet in the view.
  - Set the Visibility Applet Type property to `All`.

You must also consider the following:

- If no visibility applet type is set, the application uses the most restrictive default type, which is team-based (for example, Sales Rep).
- Do not make All, All…Across Organizations, and Admin Mode views available to mobile users. Using these views will confuse users and can lead to data integrity issues. Mobile users only have a small subset of records in their local database. A user with access to an All view could mistakenly assume that this is a complete list of all records, which it is not; it is limited to records that exist in the user's local extract. One of the potential consequences from this is data corruption. For example, a Primary Id can be reset from a valid value to No Row Match Id if a view exposes a record and its related records do not exist in the local extract.
- In a production environment, use the Administration views to make sure the Local Access flag is set to `False`. When you are in a testing environment, you can set it to `True`.
- An Administration view displays all the records in the database by disregarding any visibility properties. You configure this in Siebel Tools by setting the Admin Mode Flag to `True` and leaving the Visibility Applet and Visibility Applet Type properties blank.
- Some data in your application will be visible with no constraints. This includes enterprise-visible data, such as tables that have a Docking Visibility Type of Enterprise.

Docking Visibility and Dock Objects

To access server data, you must be either a connected or a mobile user. A connected user views data from the server database through a local or wide area network. A mobile user downloads database records to a laptop through the synchronization process. The previously described visibility rules apply to both connected and mobile users. However, for mobile users, an additional category of restrictions, called Docking or Routing Visibility, is placed on user access to records.

NOTE: Routing restrictions on data access apply *in addition* to visibility restrictions.

The docking visibility rules determine which records from the server database are delivered to each mobile user. Docking visibility rules can be viewed but not edited in Siebel Tools (except by Siebel Expert Services and even then, only in specific situations). Although you cannot change these hard-coded visibility rules, in some situations you can add docking rules. Sometimes you may have to identify and interpret docking rules, as in the following situations:

- When investigating the relationships and rules to determine which records will be routed to the mobile user either during design or troubleshooting problems.
- When deciding whether to use a particular table in the Siebel data model for a nonstandard purpose. In this case, it is critical that the table's intended use is consistent with its docking visibility rules.

NOTE: Docking rules are often known as synchronization or routing rules.

For detailed information on routing, you can use Siebel Tools to examine the properties of the Dock Object object type and its child object types. Each Dock Object has many Dock Object Tables and Dock Object Visibility Rule object definitions.

The Visibility Level property in a Dock object specifies at a high level whether all records will be transferred for the corresponding set of tables, or only a limited set that corresponds to combinations of Personal, Sales Team and Manager visibility. Three Visibility Level property settings are possible:

- **Enterprise.** No restriction on the transfer of records. An example of this Visibility Level property setting is External Product.
- **Private.** Used exclusively for routing of nonconfigurable data. This setting makes sure that the rows in these dock objects are never routed to any mobile clients and remain Private on the server.
- **Limited.** Specifies a distribution of records that, depending on the dock object and user logon ID, correspond to some combination of Personal, Sales Team, and Manager visibility. Party, which merged Account, Organization, Contact, and Employee dock objects from the previous release, is an example of this type of setting.

Dock Object Tables

Dock Object Tables are a logical collection of physically related database tables. For example, in addition to routing Opportunity records contained in S_OPTY, there are many related records which may also need routing. Examples of these related records could be Opportunity Attachments, Opportunity Notes, and configuration-specific data stored in S_OPTY_XM.

Dock Object Visibility Rules

Dock Object Visibility rules are used only for limited visibility objects. They determine which records are routed to the mobile user. As an example, consider the Opportunity dock object. Rules exist to route records for which the user is on the sales team for the opportunity, or for which you are the manager of an employee on the sales team, or for opportunities that are available through drilldown. However, there are many more records available. To determine the sequence in which rules are evaluated, sort the list on descending Visibility Strength and then on ascending Sequence. Once a rule evaluates to true, no further rules are evaluated.

There are seven types of Dock Object visibility rules:

- **E-type rule, Employee.** An Employee is associated to the logical record. For example, the user is an owner of a service request record.
- **F-type rule, Manager Of Employee.** The subordinate of the user (a manager) is associated to the logical record. For example, the user is the manager of another user who owns a service request record.
- **P-type rule, Position.** A position is associated to the logical record. For example, if you are on the sales team for an account, you will receive the account record.
- **Q-type rule, Manager Of Position.** The subordinate of the user (a manager) is associated to the logical record. For example, the user is the manager of another user who is the primary on an account team.
- **N-type rule, Node (Selective Account Retrieval).** Used for selective account retrieval. This visibility rule is used to restrict the list of accounts routed to individual users by transferring only the accounts each salesperson needs, instead of all accounts to which the salesperson has visibility when connected to the server database. Selective retrieval is implemented through a combination of disabling a dock object visibility rule and adding account names to mobile clients in an administrative view. Disabling a visibility rule affects all mobile users accessing the server, so when considering this type of change, consult with Siebel Expert Services first.
- **S-type rule, SQL Statement.** Routes data based on a specific SQL select statement that is defined as part of the rule. For example, the rule to route an account that has been deemed as enterprise-visible uses a SQL statement to route based on the value of the ENTERPRISE_FLG column.
- **C-type, Check Dock Object.** Evaluates a secondary object. If the object being checked is retrieved with a Visibility Strength that is greater than or equal to the Check Dock Object Strength of the rule, then the rule evaluates to true. For example, for the Organization (Account) Dock Object, the Contact object is checked to be sure that if you have visibility to a contact, you also receive the records containing the contact's account.

Visibility Strength

The final part of determining the docking of records is understanding the visibility strength property of objects. Visibility strength is a value with a range of 0-100 and it is used in two places:

- **Comparison Against Dock Object Tables.** If a docking rule evaluates to true, then the strength is compared with that of the related Dock Object Tables. If the strength of the rule is greater or equal to that of the related tables, then the user gets the associated records in those tables. To clarify this, consider a scenario where you are a mobile user and have received two opportunity records but you have only received the related notes for the first.
  - In the first scenario, you received the opportunity because you are on the sales team. This would give you a strength of 100 (also referred to as full visibility). Checking the Dock Object Tables reveals that you need a strength of 100 to receive the related notes, which in this case is true.
  - In the second scenario, you are not on the sales team. You received the opportunity because you have visibility to the related account. This time, you receive the opportunity record, but with a lower strength of 50 (also referred to as partial visibility). However, because you need a strength of 100 to receive the notes, for this record, they are not routed to you.
- **Comparison For Type C Rules**. If the object being checked is retrieved with a Visibility Strength that is greater than or equal to the Check Dock Object Strength, the rule evaluates to true. Consider a scenario where you are a mobile user and have received two opportunity records but you only received the related activities for the first opportunity.
  - In the first scenario, you received the opportunity because you are on the sales team. This would give you a visibility strength of 100 for the opportunity. Checking the Activity Dock Object reveals that you need a visibility strength of 100 to receive any related activities. In this case, the Check Dock Object rule evaluates to true and you receive the activities.
  - In the second scenario, you are not on the sales team. You received the opportunity because you have visibility to an asset that is related to the opportunity. This time, you receive the opportunity records but with a lower visibility strength of 75. The Check Dock Object on Activity evaluates to false and, based on the opportunity record you received, you do not get the activities.

Database Extensibility 10

This section describes database extensibility.

Database Extensibility Overview

Database extensibility refers to extending your database's schema. When you are developing a strategy to extend your database, consider using the different types of extensibility in the following order:

- Static database extensibility

  This is the first type of extensibility to consider when planning changes to your schema. The advantage of using this type of extensibility is that it allows you to adapt the database to suit a specific need without altering the schema in any way. This leads to the automatic use of optimizations that are built into the standard system.

  This type of extensibility involves the use of predefined columns that have no other business purpose. These columns would either be in tables that have a one-to-one relationship with an existing base table (for example, a predefined table with an S_ prefix and an _X suffix), or in tables that have a many-to-one relationship with an existing table (for example, a table that has an S_ prefix and an _XM suffix).

- Dynamic database extensibility

  This is the second type of database extensibility to consider using to modify the schema. Dynamic database extensions modify the schema in a defined way. You can use Siebel Tools to add columns through the Object List Editor or create an extension table by navigating to a Table object and clicking the Extend button. They allow you to create new columns on base tables and new extension tables, with records that have a one-to-one relationship with records in a base table.

Static Database Extensions

- **_X extension tables (1:1).** This type of extension table is basically an extension of the base table record. You do not need to create a new business component object when you use these tables. The _X tables are implicitly defined as joins.
- **_XM extension tables (1:M).** This type of extension table lets you create a one-to-many relationship with the record in the parent base table. The following list provides the guidelines to follow when using _XM tables:
  - Create a new business component based on the relevant _XM table.
  - Create a Type field based on the TYPE column and use this to filter the records by specifying it in the search specification for the business component.
  - Add the correct Predefault Value to the Type field.
  - Use the NAME column to track the main data field.
  - The unique index is on PAR_ROW_ID, TYPE, NAME, and CONFLICT_ID. Be aware of this index when planning your design.
  - If you are using MVGs to display data, extend the base table to track primary values.

Dynamic Extensions

As with all configuration work, first perform any database extensibility in a local database. This provides the ability to recover if there are any unexpected issues and allows the thorough testing of changes before they are made available to other developers. The following steps illustrate this process:

1. **Check out the project.** With Siebel Tools running in a local database, check out the project containing the table that you want to modify. Typically, this is the Newtable project.

2. **Update the logical schema definition in the local environment.** Make your changes to the requisite tables, columns, indexes, and mappings in the project. For more information, see *Siebel Tools Reference, MidMarket Edition*.

3. **Apply the physical schema extensions to the local database.** When you are finished making changes, complete the following tasks to update your local environment:

   a. In the Table List window, click Apply.

      This applies your changes to the local database. You may get a warning message that database extensibility is not supported from the client.

   b. Specify Current Query, All or Current Row. Enter your local database password (which *must* be in uppercase), and then click Apply.

   c. In the Table List window, click Activate.

      Thoroughly test any changes. Typically, you expose data model changes in screens or views, so you must test any of them that may have been affected by your changes. You must also check out any updated copies that have been modified by other developers, and test them in your local database.

4 **Apply the changes to the server database.** Check in the Newtable project in order to update the repository schema definition on the server. At this point, the logical database schema of the server database has been updated, but the changes have not been applied to the physical server database.

5 Repeat Step 3 while connected to the server using Siebel Tools.

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. For more information, see *Siebel Tools Reference, MidMarket Edition*.

Index